# *InveStore*

## PROGRAMMER'S MANUAL

Pegasus Disk Technologies, Inc.
2333 San Ramon Valley Blvd.
San Ramon, CA 94583

All rights reserved. This document, or parts thereof, may not be reproduced in any form without the express permission of Pegasus Disk Technologies.

All product names referenced herein are trademarks or registered trademarks of their respective companies.

The information contained in this document is subject to change without notice.

© 2000 Pegasus Disk Technologies, Inc.

# CONTENTS


Programmer's Manual ..... i
Introduction ..... 7
- Before You Start Programming ..... 7
  - Read Chapter 1, "API User's Guide" ..... 7
  - Learn the GUI Controls ..... 8
- About this Manual ..... 8
  - Terminology ..... 9
  - Documentation Conventions ..... 9
- Programmer Support ..... 10

API Users Guide ..... 11
- Map of VIM Calls to GUI Features ..... 12
- Building the Application ..... 15
- Header Files ..... 15
  - VIMAPI.H ..... 15
  - VIMERR.H ..... 15
  - VIM32.LIB ..... 15
  - Compiling and Linking ..... 16
  - VIM API 3.x vs. prior versions ..... 16
  - Win32 Environment ..... 17
- Committing a Volume ..... 17
  - Flushing Dirty Volumes Programmatically ..... 18
  - Flushing Dirty Volumes Using ATTRIB ..... 18
- Free Space on a Volume ..... 19
  - Free Space Using the Win32 Interface ..... 19
  - Transparent Bytes Free ..... 20
  - Total Bytes Free from MS-DOS and Windows 95/98 Clients ..... 21
  - Transparent Bytes Free Under NetWare ..... 21
- Using VIM_get_volume_info to Determine Volume Free Space ..... 22
- Volume Interface Manager Calls ..... 22
  - Backward Compatibility ..... 23
- New Function Calls ..... 23
  - Supported Calls ..... 23
- Calling Conventions ..... 26
  - System Data Types ..... 26

Current Directory Structure ..... 27
InveStore Records ..... 27
Common Fields ..... 28
Directory Record ..... 29
File Record ..... 30
Subdirectory Record ..... 31
Volume Record ..... 31
Record Object Union ..... 35
Return Codes ..... 36
Pathname Rules ..... 36
Cache Write Algorithms ..... 36
Lazy Write ..... 36
Flush on Close ..... 37
Write Through ..... 37
Optical Storage Considerations ..... 37
Lazy Write Application Suggestions ..... 38
Identifying Unflushed Volumes ..... 39
When to Use the VIMcommitVolume ..... 39
Other Info ..... 39
Optical File System Overview ..... 41
System Overview ..... 41
Volume Manager ..... 41
Volume Types ..... 42
Threads ..... 42
Optical File Server ..... 42
OHIM ..... 42
Hierarchical Cache Management ..... 43
HCM Operation ..... 43
HCM Implementation ..... 44
File Formats ..... 44
HPWOFS Rev 1 ..... 45
HPWOFS Rev 2 ..... 45
WORM ..... 46
OSTA-UDF/ECMA 167 Compliant ..... 46
Physical File Structure ..... 46
Control Area ..... 47
Data Area ..... 47
Logical File Structure ..... 48
File System Operations ..... 48
File Accessibility ..... 49
System Software ..... 51

InveStore Kernel ..... 52
Volume Interface Manager ..... 53
Logical File Server ..... 54
Physical Format Manager ..... 54
Hierarchical Cache Management ..... 54
BTREE and Directory Cache Manager ..... 54
Physical File Server ..... 55
Memory Manager ..... 55
User-Configured System Parameters ..... 55
General Parameters ..... 56
Cache Parameters ..... 56
Mount Parameters ..... 58
Disk Swapping Parameters ..... 59
InveStore Initialization ..... 63
API Reference Guide ..... 65
VIM Interface Calls ..... 65
VIM_chdir ..... 65
VIM_check_eof ..... 66
VIM_cleanup_process ..... 66
VIM_close ..... 67
VIMcommitVolume ..... 67
VIM_create ..... 68
VIM_delete ..... 69
VIM_disk_params ..... 70
VIM_flush_buffers ..... 71
VIM_format ..... 71
VIM_format_media ..... 74
VIM_fseek ..... 76
VIM_get_drive_bias ..... 77
VIM_get_file_info ..... 78
VIM_get_hardware_list ..... 78
VIMgetHardwareListSize ..... 85
VIM_get_object_attributes ..... 86
VIM_get_object_info ..... 86
VIM_get_time_date ..... 87
VIM_get_volume_info ..... 88
VIM_initialize ..... 89
VIM_is_valid_path ..... 89
VIM_mkdir ..... 90
VIM_mount ..... 90
VIM_open ..... 92

VIMopenCreate ..... 94
VIM_read ..... 97
VIM_rename ..... 98
VIM_rmdir ..... 99
VIM_search ..... 100
VIM_set_drive_bias ..... 101
VIM_set_drive_status ..... 103
VIM_set_object_attributes ..... 103
VIM_set_time_date ..... 104
VIM_set_volume_attr ..... 105
VIM_set_volume_info ..... 106
VIM_shutdown ..... 107
VIM_truncate_file ..... 107
VIM_unmount ..... 108
VIM_write ..... 109
VIM Programming Examples ..... 110
Error Codes ..... 115
General Format of the Error Log File ..... 116
Fatal Errors ..... 116
Error Code Directory ..... 117
General Errors ..... 117
Memory Manager Errors ..... 128
Btree Errors ..... 130
File Allocation Descriptor (FAD) BTREE Errors ..... 140
PFM error codes ..... 142
Mount Error ..... 143
OHIM Errors ..... 144
DLL Errors ..... 156
Basic Disk Errors ..... 157
Asynchronous Timer Errors ..... 157
Thread/Process Errors ..... 158
Semaphore Manager Errors ..... 159
HCM Errors ..... 161
Interprocess Communications Errors ..... 165
File Errors ..... 166
Shared Memory Errors ..... 166
Troubleshooting ..... 169
Have Your Information Ready ..... 170
Frequently Asked Questions ..... 170
Common Problems ..... 175

# INTRODUCTION

This guide outlines the InveStore programmatic interface that can be used by developers to build an application using Pegasus storage management functionality. Topics covered include:

- How to use the Application Programming Interface (API)
- How the optical file system works
- The InveStore software modules
- The API access points

## Before You Start Programming

Before you start using the API, take the time to familiarize yourself with the API basic concepts and standard InveStore GUI controls. This will help you avoid mistakes that might occur if you don't understand the basics of the software package.

### Read Chapter 1, "API User's Guide"

This chapter contains the basic information you need to know before trying to use the VIM API. You *must* read this chapter to master the VIM API.

## Learn the GUI Controls

Read the User's manual. The GUI uses the VIM API to activate most of the menu items. For more information, see "Map of VIM Calls to GUI Features" on page 12. Understanding how the GUI works and the limitations on certain commands will help you better understand how to use the VIM API.

# About this Manual

This manual is organized as follows:

Chapter 1, "API Users Guide", describes how to use the InveStore API.

Chapter 2, "Optical File System Overview", describes the InveStore optical system.

Chapter 3, "System Software", describes the InveStore software.

Chapter 4, "API Reference Guide", provides detail reference material for the InveStore API.

Appendix A "Error Codes," documents the error codes users may encounter.

Appendix B "Troubleshooting," describes techniques for isolating problems.

## Terminology

The following terms used in this manual are important to understand:

| | |
|---|---|
| **optical sub-system** | The underlying software and hardware for optical media management. |
| **optical disk** | The generic name for the medium that stores data. InveStore software supports WORM, rewritable and CD-ROM and DVD-ROM media. |
| **OFS** | Optical File System - a generic name for InveStore and its underlying file system(s). |
| **volume** | 1) The physical cartridge that you load into a jukebox. Used interchangeably with cartridge, disk cartridge, optical disk and CD/DVD-ROM disk.<br><br>2) One side of an optical disk, unless you are referring to a dual-headed optical drive that views both sides of a disk as one volume. Used interchangeably with disk volume and optical disk volume. |
| **optical drive** | A device that reads optical disks. |
| **bias** | The read/write disposition of an optical drive. |
| **jukebox** | One or more drives, storage space for cartridges, and an autochanger. |
| **component** | An installable optical drive that is fully supported by Pegasus. |
| **transparent access** | Using a drive letter as well as the ability to use standard operating system calls to interact with the optical system. |

## Documentation Conventions

The following conventions are used in this manual:

| **Convention** | **Meaning** |
|---|---|
| *Italic* | A new or important term. Also indicates a variable, which you must supply, such as a file name. |
| `Courier font` | Directory names, parameter names, and source code examples. |

Identifies those features that do not apply to CD/DVD-ROM systems

# Programmer Support

For API related questions visit our online eSupport search engine at http://www.pegasus-ofs.com/apisupport.

Basic assistance with API questions is provided to ISVs who are investigating the Pegasus API. In addition, registered software users can receive direct help if they have an active annual maintenance contract or are within the initial 90-day warranty/support period.

**Note** It will be assumed that you have the necessary training and background in programming, and that you have previously reviewed this guide and the VIMTEST information provided with your copy of InveStore.

You will need to provide the following information so that our Engineering department can review your problem:

- A detailed description of what you were attempting when the error occurred, including a print out of the actual call you were making.
- The name of the VIM call.
- Error messages received. (Include any Application errors and what module was running when the error occurred.)
- A copy of the Dr. Watson Log (\winnt\drwtsn32.log) if your call generated this type of error.
- The InveStore system LOG files and PSUPPORT information.
- The compilable source for analysis.

In most cases our Engineering Department will need to be able to look at your code to be able to determine why the process failed.

If your question or problem requires application “debugging” or programming development, it will be necessary for you to contract for Pegasus Engineering Services. There is a $195 per hour charge for this level of engineering service.

E-mail can be sent to: support@pegasus-ofs.com

# 1 API USERS GUIDE

The VIM (Volume Interface Module) is the Optical File Server Protocol accessed as a group of C function calls contained within a DLL. The VIM is a 32-bit interface (16-bit calls are not supported).

This chapter describes the following topics:

- How VIM calls map to GUI features
- Header files
- Instructions for building your application
- Win32 programming information
- Information that is common to many of the calls, such as parameters and field attribute format
- A brief description of the VIM calls
- A table mapping the VIM calls to actual implementations in InveStore
- How to use the cache

**Note** The Pegasus VIM interface is subject to additions and modifications without notice. New calls may be added as needed.

Always obtain the latest header files and full source code examples from the \PDT\VIM subdirectory after installation. The information you will find there contains the latest call enhancements to the VIM interface and fixes for inaccuracies in this manual.

Verify prior to installing a new release that any changes to the VIM interface do not affect your existing programs.

# Map of VIM Calls to GUI Features

The following table shows you the VIM calls that Pegasus Disk Technologies used to implement certain functionality in InveStore. These implementations serve as examples of how to use the VIM calls:

| GUI Feature | VIM call | Function / Comments |
|---|---|---|
| Change library name | None | The Library name is internal to the GUI and not supported by the API. |
| Commit volume | VIMcommitVolume | The GUI uses this call to ensure a given volume has all the data committed. |
| Assign logical drive | None | This feature is *not* implemented using a VIM API call. The GUI uses the operating system to assign drive letters. The API does not use drive letters. |
| Preload Initialize | VIM_format | The Preload Initialize function. The GUI uses VIM_format to format any trapped volume. This is the main purpose of VIM_format and includes formatting !INI, !ERR, !PDT, and any mounted InveStore volume that is no longer needed. The capability to reformat mounted volumes is new for version 3.0. This call only formats media that is inside a drive or jukebox. It does not take media from the mailslot. Use the Eject After Format button and the MountAfterFormat parameter if you want to take media from the mailslot; Setting MountAfterFormat to 0 causes the media to be ejected. |
| Format a volume | VIM_format | In this release, you can automate formatting trapped volumes, (files prefixed by !*xxx*) or an existing InveStore file system. No human intervention is required. In the previous release, you had to export the disk, delete cache, and use VIM_format_media to format over an existing volume, which you had to bring in through the mailslot. |

| GUI Feature | VIM call | Function/Comments |
|---|---|---|
| Initialize a volume | VIM_format_media | Initialize/Reinitialize function. The GUI uses this command for importing media from the mailslot. The Radio button's Eject Upon Completion is implemented using the MountAfterFormat Parameter. The Initialize /Reinitialize button uses the Reformat parameter. |
| Edit menu | VIMgetHardwareListSize | The settings displayed when the user selects Library/Drive Settings from the Edit menu on the main console window. |
| Edit menu | VIM_get_hardware_list and VIM_search | The information on mounted and unmounted volumes that is displayed in the main console window. Library\Drive Settings information is obtained from VIM_get_hardware_list. |
| Details button | VIM_get_object_info and VIM_search | The file volume information displayed when the user clicks the Volume Details button in the Library/Drive Settings window. |
| Change drives | VIM_get_drive_bias | Information displayed when the user clicks the Drive Details button in the Library/Drive Settings window. The GUI gets the drive bias using VIM_get_drive_bias and sets it using VIM_set_drive_bias. For more information on drive bias, see "VIM_get_drive_bias" on page 77. |
| Get reserved space setting | VIM_search, VIM_get_object_info, or VIM_get_volume_info | Retrieving the current reserved space setting from the volume record. |
| Mount volume | VIM_mount | Importing a volume. The GUI uses this command to import a new disk either from the mailslot or from a stand-alone drive. The GUI does not expose the mount_type parameter. |
| Rename volume | VIM_rename | Renaming a volume by selecting Rename Volume from the File menu. The GUI uses the standard rename command to rename volumes. |

| GUI Feature | VIM call | Function / Comments |
|---|---|---|
| Refresh | VIM_search | The display resulting from selecting Refresh on the File menu. The GUI simply calls VIM_search with the initial search pattern set to "*.*". This finds everything in the root of the file system. All root entries are volume records. The GUI then adds to the Mounted Volumes window, any new volume records that finds on line. |
| Change drives | VIM_set_drive_bias | .The GUI gets the drive bias using VIM_get_drive_bias and sets it using VIM_set_drive_bias. For more information on drive bias, see “VIM_set_drive_bias” on page 101. |
| Change drives | VIM_set_drive_status | Activates online and offline radio buttons, which are set in the Drive Details window that is accessed from the Library / Drive Settings window. |
| Set reserved space | VIM_set_volume_info | Change reserved space settings. The Reserved Space Settings menu item on the Edit menu is used to reserve a number of sectors on a volume. |
| Shutdown optical system | VIM_shutdown | Shuts down the optical system. |
| Unmount volume | VIM_unmount | Exports media from the system, often sending it to storage. |

# Building the Application

This section covers the requirements for compiling and linking an application that uses the VIM interface.

# Header Files

The first consideration for any application is the inclusion of the VIM API header files in your source code. The files define the basic structures, function prototypes, and return codes that are needed to use the VIM API.

## VIMAPI.H

VIMAPI.H is a file to be included in C programs to provide definitions for the VIM interface to InveStore. For Windows NT/2000, it is developed and tested using Microsoft Visual C/C++. If another compiler is used, minor changes may be required. This file must be added to the customer's programming environment in a way that allows the compiler to locate it. Placing it in the same directory as the source file which includes it will suffice.

## VIMERR.H

VIMERR.H is another include file which contains only error code definitions. It should be included in any application which processes error returns from InveStore so that they may be referenced symbolically.

## VIM32.LIB

VIM32.LIB is an import library containing definitions of the public entry points to InveStore. It must be provided to the application during linking. The calling sequence type is _stdcall. This is declared in the function prototypes, so the application need not use _stdcall by default.

## Compiling and Linking

The following example shows compile and link commands for Visual C:

```
CL /c /MT /Zp1 program.c
LINK program.obj VIM32.LIB /OUT:program.exe
```

The compiler arguments are:

| | |
|---|---|
| **/c** | compile only |
| **/MT** | multithreading (if desired) |
| **/Zp1** | pack structures on byte boundaries (a must!) |

## VIM API 3.x vs. prior versions

For InveStore, the calling convention for the VIM API uses _stdcall to help support developers that use other languages such as Visual Basic to access the API. Prior versions used the _cdecl calling convention. This fact requires that you recompile and relink any code written to interface with older Pegasus software versions.

The format of the data record returned for file and directories has changed. The GENERIC_REC is now used for both file and directories. The separate file_rec and subdir_rec are no longer used. It is suggested that you familiarize yourself with the structure of GENERIC_REC. This change affects the following calls:

- VIM_get_object_info()
- VIM_get_file_info()
- VIM_search()

If you use any of these function calls, you must change the code to use the new record structure. We recommend that you familiarize yourself with the new record prior to coding.

## Win32 Environment

Use Visual C++ to build your Win32 applications. Use the following options:

- Use byte packing on structures when calling the VIM interface.
- _stdcall option for public interfaces.

### Include Files

Ensure that the compiler can find include files. For example, use the /I option to specify the search directory for include files.

### Library Files

Ensure that the linker can find the library files. See the following example.

### Example

This example makes the following assumptions:

- The PATH environment variable is set to the directory that holds the compiler executables, for example *Drive*:\MSDEV\BIN.
- You installed InveStore in the default directory, *Drive*:\PDT.
- You created a subdirectory for API header files named *Drive*:\PDT\INCLUDE.
- You copied vimapi.h and vimerr.h to *Drive*:\PDT\INCLUDE.
- You have vim32.lib in the *Drive*:\PDT\DLL directory.

Using Visual C++, the following commands create an executable called vsearch from a source file called vsearch.c:

```
SET LIB=Drive:\PDT\DLL;%LIB%
SET INCLUDE=Drive:\PDT\INCLUDE;%INCLUDE%
CL.EXE /c /Zpl Progname.C
LINK Progname.obj VIM32.LIB /MAP:Progname.map /OUT:Progname.exe
```

# Committing a Volume

This section discusses how to commit data from the InveStore memory cache to an optical volume.

## Flushing Dirty Volumes Programmatically

If you're programming on a workstation over the network, you *cannot* use the VIM interface to commit a dirty volume. Instead, use the Win32 interface.

**Note** A dirty volume is data in RAM cache that has not yet been written to optical.

The Win32 call is an alternative to calling the new VIMcommitVolume function directly. For more information on VIMcommitVolume, see "VIMcommitVolume" on page 67.

The following code fragment shows how to find dirty volumes and commit them:

```
DWORD Attributes;
Attributes = GetFileAttributes (<Drive:\\Volume>);
if (Attributes & FILE_ATTRIBUTE_ARCHIVE)
   {
    Attributes &=  ~FILE_ATTRIBUTE_ARCHIVE;
    SetFileAttributes (<Drive:\\Volume>);
    }
```

In the example above, clearing the archive flushes all the volume's uncommitted data, including directory information, to optical.

In the example above, *<Drive:\\Volume>* represents a text string with the drive specifier. For example, to call GetFileAttributes on a file named MYVOLUME on the O drive, use the following syntax:

```
DWORD Attributes =
   GetfileAttributes ("O:\\MYVOLUME");
```

**Note** Using the ATTRIB command is equivalent to calling SetFileAttributes.

## Flushing Dirty Volumes Using ATTRIB

The standard Operating System ATTRIB command, which is used to get and set file and directory attributes, may be used for determining which volume contains uncommitted data.

The ATTRIB command may also be used to force any volume that contains uncommitted data and directories to be flushed to optical.

To identify dirty volumes, use the following syntax:

```
ATTRIB O:\*.*
```

The output is:

```
A  O:\MYVOLUME.1
   O:\MYVOLUME.2
```

In the output, an A to the left of the volume name, identifies a dirty volume.

### Finding Dirty Volumes with the Archive bit

The archive bit is set on any volume that contains uncommitted data. An application can programmatically check this bit and clear the archive. InveStore automatically sets the archive bit on any volume that contains uncommitted data.

Using ATTRIB on the root of InveStore shows you which volumes have the archive bit set, and therefore have not been committed to optical.

In the last Example, MYVOLUME.1 has the archive bit set as indicated by the A.

To clear the archive bit, which flushes all data for MYVOLUME.1 to optical, use the following syntax:

```
ATTRIB –a O:\MYVOLUME.1
```

# Free Space on a Volume

Using InveStore, you can use any one of several ways to determine the amount of free space available on an optical volume. This section reviews each method.

## Free Space Using the Win32 Interface

You can determine the amount of free space on a volume within InveStore by calling the Win32 interface as shown here:

```
DWORD dwBytesFree;
HANDLE hFind;
WIN32_FIND_DATA WFD;
hFind = FindFirstFile ( "drive:\\volume", &WFD );
if( hFind!=INVALID_FILE_HANDLE ){
   FindClose( hFine );
   dwBytesFree = WFD.nFileSizeLow;
}
```

This example assumes *drive* is assigned to the location of InveStore and *volume* is the name of a volume within InveStore. Only volumes that are smaller than 4 Gb are supported by this feature.

Another method uses call “GetDiskFreeSpaceEx”. To find available space on a given volume, your application must properly fill in the volume specification parameter with “driveletter:\volumename\anexisting-directory”.

For example:

```
Spec = “O:\MYVOLUME\ABC123
IResult = GetDiskFreeSpaceEx(Spec, liAvailable,liTotal,liFree);
```

## Transparent Bytes Free

The current release of InveStore supports obtaining bytes free using a standard programming calls with some caveats. To obtain free disk space for a particular volume, you must first issue a FindFirst call with the drive letter and volume name for the volume that you wish to obtain the bytes free information. This find first information is required by InveStore to correctly identify for which volume the free disk space call is intended. The following example uses a hard coded drive D:\ as the optical server and OpticalVolume as the name of the volume for which free space is needed.

```
// Variables for processing the disk space information
DWORD            dwSectorsPerCluster, dwBytesPerSector;
DWORD            dwFreeClusters, dwClusters;
DWORD            dwBytesFree;
HANDLE           hFind;
WIN32_FIND_DATA  WFD;
DWORD            FreeBytes;

hFind = FindFirstFile ("d:\\OpticalVolume", &WFD);
if (hFind!=INVALID_FILE_HANDLE)
{
    FindClose (hFine);
    dwBytesFree = WFD.nFileSizeLow; // Not transparent but info is here

   // Get the disk space information.

   if  (GetDiskFreeSpace ("D:\\",   &dwSectorsPerCluster,
        &dwBytesPerSector, &dwFreeClusters, &dwClusters))
       {
         FreeBytes = dwSectorsPerCluster * dwBytesPerSector *
                     dwFreeClusers;
       }

   }
```

```
   else
   {
     // Error;
   }
}
else
{
   // Error;
}
```

Generally speaking, the only reason to use the transparent method is when a product will be dealing with multiple file systems. In this case, you can perform the FindFirst against all file systems using the path you are writing to as input for the FindFirst. This will have no negative effects for the other file systems used, but will allow the GetDiskFreeSpace to work as expected with InveStore.

## Total Bytes Free from MS-DOS and Windows 95/98 Clients

Obtaining free disk space in a transparent fashion is currently not available when the InveStore resource is shared using Microsoft's File and Print Services for NetWare. The following code is the only method available under this environment to obtain free space information/

```
_
DWORD               dwBytesFree;
HANDLE              hFind;
WIN32_FIND_DATA     WFD;

hFind = FindFirstFile ("drive:\\volume", &WFD);

if (hFind!=INVALID_FILE_HANDLE)
{
    FindClose (hFine);
    dwBytesFree = WFD.nFileSizeLow; // Not transparent but info is here
                    }
```

## Transparent Bytes Free Under NetWare

From an NT Client under NetWare, volumes with more than two gigabytes of free space will show half as much free space as they actually have. The following code is the only method of obtaining accurate free space information.

```
_
DWORD               dwBytesFree;
HANDLE              hFind;
WIN32_FIND_DATA     WFD;

hFind = FindFirstFile ("drive:\\volume", &WFD);

if (hFind!=INVALID_FILE_HANDLE)
{
```

```
    FindClose (hFine);
    dwBytesFree = WFD.nFileSizeLow; // Not transparent but info is here
}
```

## Using VIM_get_volume_info to Determine Volume Free Space

The InveStore VIM call, VIM_get_volume_info, discussed in Chapter 4, returns information about a specific volume and is most often used in determining volume free space.

**Syntax**

```
int ENTRY VIM_get_volume_info
(pTEXT volume_name,
pUSHORT sector_size,
pULONG total_sectors,
pULONG free_sectors,
pULONG user_reserved,
pULONG system_reserved)
```

**Input**

**volume_name** - The name of the volume for which you would like information.

**Output**

**sector_size** - Contains the physical sector size of the volume.
**total_sectors** - Contains the total number of sectors for this volume.
**free_sectors** - Contains the number of available sectors on the volume. **Available** in this case means **unwritten** sectors and includes *user_reserved* and *system_reserved* sectors.
**user_reserved** - The number of sectors that have been reserved by the user.
**system_reserved** - The number of sectors reserved by InveStore for internal bookkeeping.

The VIM call returns the actual number of free sectors as well as the user reserve and the system reserve. To find the bytes free, subtract *user_reserved* and *system_reserved* from *free_sectors* and multiply by your *sector_size*.

# Volume Interface Manager Calls

The following table contains a brief description the Volume Interface Manager calls. For more information, see Chapter 3, “Programmatic Interfaces to InveStore.”

## Backward Compatibility

Applications based on previous releases of the API are compatible with this release. Recompile and relink the existing application with the new DLLs —VIM32.DLL and VIMSYS.DLL.

# New Function Calls

There are two new API calls that you may wish to incorporate into your application:

- VIMopenCreate
- VIMcommitVolume

VIMopenCreate allows you to control caching on a per file basis. VIMcommitVolume allows your application to ensure that the cache is flushed to optical.

## Supported Calls

The following table lists all calls that are supported by the InveStore API.

**Supported Calls**

| | *Function* | *Description* |
|---|---|---|
| | VIM_chdir | Changes the current directory. |
| | VIM_check_eof | Checks the specified handle to determine if the file is at end-of-file condition |
| | VIM_cleanup_process | Cleans up all resources associated with a process when it terminates. |
| | VIM_close | Closes the file associated with the supplied file handle. |
| | VIMcommitVolume | Saves the data in cache to the optical disk. |
| | VIM_create | Creates a new file or overwrites an existing file. |

**Supported Calls (*continued*)**

| | *Function* | *Description* |
|---|---|---|
| CD | VIM_delete | Deletes the file specified by file name. |
| | VIM_disk_params | Supplies disk information for a given volume. |
| | VIM_flush_buffers | Flushes and synchronizes the DASD cache buffers. |
| CD | VIM_format | Formats a disk cartridge. |
| CD | VIM_format_media | Formats a disk cartridge. |
| | VIM_fseek | Changes the position of the file pointer. |
| | VIM_get_drive_bias | Determines the bias of the requested drive. In the CD/DVD-ROM software, this function is always set to None. |
| | VIM_get_file_info | Returns the file record for the file associated with the open file handle “handle.” |
| | VIM_get_hardware_list | Gets a list of hardware information. |
| | VIMgetHardwareListSize | Determines the size of the packet containing the list of hardware information. |
| | VIM_get_object_attributes | Retrieves the attributes for a volume, subdirectory, or file object. |
| | VIM_get_object_info | Returns the file, subdirectory, or volume control record. |
| | VIM_get_time_date | Returns the time and date stamp for the file based on its open handle. |
| | VIMopenCreate | Opens a file and defines caching. |
| | VIM_get_volume_info | Returns information about the desired volume. |
| | VIM_initialize | Initializes the communications channel with the optical file server. |
| | VIM_is_valid_path | Determines if a given path is valid or invalid. |

**Supported Calls (***continued***)**

| | *Function* | *Description* |
|---|---|---|
| CD | VIM_mkdir | Makes a new directory. |
| | VIM_mount | Imports new volumes into the system. |
| | VIM_open | Creates/opens the requested file. |
| | VIM_read | Reads data from the open file. |
| | VIM_rename | Renames files, subdirectories, and volumes, including CD/DVD-ROM disks. When you rename a CD/DVD-ROM, the renaming occurs in cache. When the case is removed, the disk reverts to the original volume name. |
| CD | VIM_rmdir | Removes a directory or volume. |
| | VIM_search | Performs directory search operations. |
| CD | VIM_set_drive_bias | Sets the drive bias. |
| | VIM_set_drive_status | Sets the status of a drive (on-line, off-line). |
| CD | VIM_set_object_attributes | Changes the attributes of a file, subdirectory, or volume object. |
| CD | VIM_set_time_date | Changes the time/date stamp of the file associated with the handle. |
| | VIM_set_volume_attr | Changes the extended attributes of the volume object. |
| CD | VIM_set_volume_info | Sets the number of reserved sectors. |
| | VIM_shutdown | Brings kernel to an orderly shutdown. |
| CD | VIM_truncate_file | Truncates the size of the file at the current file pointer position. |

**Supported Calls (*continued*)**

| | *Function* | *Description* |
|---|---|---|
| | VIM_unmount | Exports volumes from the system. |
| | VIM_write | Writes data to the optical disk. |

# Calling Conventions

On the NT/2000 platform, example code and DLLs are compiled from Microsoft using Visual C++. The normal calling convention for the public entry points of Pegasus DLLs is:

```
#define _stdcall
```

On the OS/2 platform, example code and DLLs are compiled with software that supports _System. The new calling convention for the _System linkage with VIM32.DLL and VIMSYS.DLL is valid for most compilers:

```
#define ENTRY _System
```

## System Data Types

The structures in this chapter use the defines shown below.

```
typedef unsigned short DID;         /* device id */
#define PTR *                     /* type pointer */
typedef long IOSIZE;   /* extended disk I/O size */
typedef IOSIZE PTR pIOSIZE;
typedef IOSIZE * *ppIOSIZE;
typedef char TEXT;
typedef TEXT PTR pTEXT;
typedef TEXT pPTR ppTEXT;
typedef unsigned long ULONG;

/* unsigned longs and ptrs */

typedef ULONG PTR pULONG;
typedef ULONG pPTR ppULONG;
typedef unsigned long ATTRIBUTE32;


/* 32 bit file attribute */
typedef ATTRIBUTE32 PTR pATTRIBUTE32;
typedef ATTRIBUTE32 pPTR ppATTRIBUTE32;
```

```
typedef unsigned short USHORT;
typedef USHORT PTR pUSHORT;
typedef USHORT pPTR ppUSHORT;
typedef unsigned char UCHAR;
typedef UCHAR PTR pUCHAR;
typedef UCHAR pPTR ppUCHAR;
typedef unsigned int UINT;


/* unsigned natural integer */
typedef UINT PTR pUINT;
typedef UINT pPTR ppUINT
typedef int NINT;


/* natural integer */
typedef NINT PTR pNINT;
typedef NINT pPTR ppNINT;
```

# Current Directory Structure

The Pegasus kernel does not keep track of the current directory for each process. The current directory information must be passed in for each call where the current directory information may be relevant. The current directory information is initialized by calling the VIM_chdir module.

The caller does not need to have current directory. Setting the volume_id field to -1 and the subdir_id field to 0 makes all calls relative to the root of the optical file system. This requires that full path information be provided for all file and directory names.

```
typedef struct
{
   SUBDIR subdir_id;      /* subdirectory number */
   VID volume_id;          /* Disk Volume number */
   UINT32 reserved;
   UINT16 reserved2;
} CurDirInfo;
```

# InveStore Records

InveStore supports a variety of different record objects. Various InveStore calls return different types of records based on the input to that call. The record types for this release of InveStore are: directory and volume. The directory record, DIR_ENTRY_DESC, replaced the file and subdirectory records of previous releases.

## Common Fields

Both records have the file attribute and creation date and time formatted the same way.

### Attribute Field

The following table lists the possible values for the attribute field of a record returned by an OFS call:

| *Bit Field* | *Attribute* |
|---|---|
| 0 | Read Only. The file cannot be deleted or updated. This field retains this meaning under any operating system. |
| 1 | Hidden File. The file is hidden from normal directory searches. |
| 2 | Operating System File. The file is hidden from normal searches. |
| 3 | Volume Label. This allows a user name to be placed on a file system volume. Entry lives in the root directory only. If this bit is used for a search parameter, the search is exclusive. This field retains this meaning under any operating system. |
| 4 | Subdirectory. Indicates that this file is a special subdirectory entry, excluded from normal searches. This field retains its meaning under any operating system. |
| 5 | Archive bit. |
| 6 - 7 | Wildcard. This has no special meaning to PDT software, but it can be used by the host system to define any attribute necessary. |
| 8 | Normal. Indicates an ordinary file with no special properties. Generally used for text, data, and applications programs. This field retains this meaning across all operating systems. |
| 9 - 15 | Wildcard. This has no special meaning to PDT (Pegasus Development) software and can be used by the host system to define any attribute necessary. |

### Time Field

The following table lists the possible values for the time field of a record returned by an OFS call

| *Bit Field* | *Attribute* |
|---|---|
| 0 - 04 | Number of 2-second increments (0-22). |
| 5 - 10 | Minutes (0-59). |
| 11 - 15 | Hours (0-23) |

### Date Field

The following table lists the possible values for the date field of a record returned by an OFS call

| *Bit Field* | *Attribute* |
|---|---|
| 0 - 04 | Day of month (1-31). |
| 5 - 10 | Month (1-12). |
| 11 - 15 | Year (relative to 1980) |

## Directory Record

The DIR_ENTRY_DESC data type represents the following structure:

```
typedef struct{
   UINT16 Length;                          // length of this structure
   VID Volume;                        // Volume containing this record
   SECTOR Sector,                     // Sector containing this record
          Twin;                 // Sector containing this record's twin
   UINT16 Offset;           // Sector Offset of this record on optical
   UINT16 Dirty;                    // == 1 iff Changed or new, else 0
   SUBDIR Parent;                          // subdir number of parent
   UINT16 NameSize;                        // length of name in bytes
   UINT16 Reserved
   ATTRIBUTE32 Attrib;                        // word wide attribute
   UINT16 CreateTime;             // encoded time of create DOS format
   UINT16 CreateDate;             // encoded date of create DOS format
   UINT32 Size;                                  // byte size of file
   UINT32 SizeH;                          // High 32 bits of file size
   SUBTYPE Type;                                 // subtype of record
   UINT16 Ver;                                      // version number
   UINT16 ModifyTime;       // encoded time of last mod in DOS format
```

```
   UINT16 ModifyDate;       // encoded date of last mod in DOS format
   UINT16 Reserved2;                              // byte size of file
   UINT16 SubTypeLength;    // length of subtype-specific information
   UINT32 DatastreamCount;                    // number of datastreams
   BYTE Reserved[48];                             // byte size of file
   UINT16 PrevMember;     // Volume member number of previous version
   UINT32 PrevSector;     // Sector member number of previous version
   UINT16 PrevOffset;            // Sector Offset of previous version
   //The following new time and date values are expressed
   //as the number of seconds since January 1, 1970 12:00 am
   UINT32 CreateTimeDate;         // 64 bit encouded time of creation
   UINT32 CreateTimeDateH;           // high 32 bits of encouded time
   UINT32 ModifyTimeDate;      //Last Time Data for file was modified
   UINT32 ModifyTimeDateH;        // high 32 bits of modify time-date
   UINT32 AttributeTimeDate; // Time that file attribute info changed
   UINT32 AttributeTimeDateH;       // high 32 bits of file time/date
   UINT32 TotalObjectSize;   // include size of all streams including
                                                      // default stream
   UINT32 TotalObjectSizeH;            // include size of all streams
                                                      // (high 32 bits)
   UINT32 RecordedSize;  // includes only recorded extents not sparse
   UINT32 RecordedSizeH;                    // High 32 bits of stream
   UINT32 ObjectID;                      // Unique ID for this object
   UINT32 ObjectIDH;        // Unique ID for this object High 32 bits
   UINT16 NumAccesses;    // Number of times object has been accessed
   UINT16 FileLinkCount;// Defines the number of symbolic or hard links
   CHAR Name[1];                     // First byte of additional stuff
   CHAR pad[327];                        // Max extra for declarations
// Additional data area, used for name, Subtype, and Datastream
// information
} DIR_ENTRY_DESC;
```

## File Record

In this release, the directory record, DIR_ENTRY_DESC, replaces the file record. The file record is returned for all file objects, which are defined as objects that contain data. The file record describes the file object.

```
typedef struct
{
   USHORT reserve1;
   char name [20];                /* asciiz name */
   ATTRIBUTE16 attribute  /* word wide attribute */
   USHORT time;        /* encoded time of create */
   USHORT date;        /* encoded date of create */
   ULONG size;             /* byte size of file */
   USHORT reserve2;
   USHORT version;            /* version number */
```

```
   USHORT reserved3[7];
} file_rec;
```

| | |
|---|---|
| `name` | The ASCIIZ name of the file. |
| `attribute` | The file attribute word. |
| `time` | Encodes the file creation time. |
| `date` | Encodes the file creation date. |
| `version` | Contains the file version number. This field is encoded in reverse order; version one of the file is equal to 0xffff (-1). |

## Subdirectory Record

In this release, the directory record, DIR_ENTRY_DESC, replaces the subdirectory record. The subdirectory record describes directory objects.

```
typedef struct
{
   USHORT reserve1;
   ATTRIBUTE16 attribute; /* word wide attribute */
   USHORT time;        /* encoded time of create */
   USHORT date;        /* encoded date of create */
   USHORT reserve2[3];
   USHORT version;             /* version number */
   char name[1];
} subdir_rec;
```

| | |
|---|---|
| `name` | The ASCIIZ path name of the subdirectory. |
| `attribute` | The directory attribute word. |
| `time` | Encodes the subdirectory creation time. |
| `date` | Encodes the subdirectory creation date. |
| `version` | Contains the subdirectory version number. This field is encoded in reverse order; version one of the file is equal to 0xffff (-1). |

## Volume Record

The volume record describes the volume associated with a particular disk cartridge. Double-sided media has two volumes associated with each disk cartridge, one per disk side.

```
typedef struct TagVOLUME_REC
{
   VID VirtualVid,                    /* volume ID of this volume */
   RootVid,      /* root VID of this member to be used for spanning*/
/* The logical OHIM storage slot can be used to determine what shelf
a disk is in. There are two Slot ID's per double-sided disk; for
example Slot Id's0 and 1 would point to slot # 0 */
   OHIMslotID;
   UINT16Member,                /* 0 if root – currently always 0 */
   CurrentVID,                /* VID of Member volume to allocate */
   /*new sectors from (Root only)*/
   VolumesInSet;                 /* Number of volumes in volume set.
                                          current limit is 1 */
   BYTE Reserved4[4];                        /* UNUSED four bytes */
   char VolumeName[256];                        /* name of volume */
   ATTRIBUTE32 Attributes;                    /* volume attributes*/
   ATTRIBUTE32 Privileges;   /* volume access privileges, not used */
   UINT16 time, date; /                                * temporary*/
   FILE_SYSTEM_TYPE
   FileSystemType;            /* Value must be a FILE_SYSTEM_TYPE */
   UCHAR MediaType;                             /* type of media*/
   BYTE Reserved1;                               /* unused byte*/
   UINT32 CreateTimeDate;                     /* time of creation */
   BOOL32 IsAccessible;          /* OK for DIRS, or Platter Access */
   UINT16 LocationStatus;                /* online/nearline/offline*/
   UINT16 SectorSize;       /* sector size of volume or block size */
   SECTOR MaxSector,                /* last sector numberon volume */
   ReservedSectors; /* # of sectors reserved defines user reserved */
   char AliasName[20]; /* alternate volume name - not currently used */
   UINT32 DiscID;                               /* unique disk id */
   char DiscName[20];                         /* unique disk name */
   char OfflineLocation[44];       /* optional offline description */
   char ExtendedAttribute[44];   /* user definable extended attrib */
   BOOL32 Dirty;               /* 1 iff record needs to be flushed */
   UINT32 MaxSubdirNumber,          /* Last subdir number assigned */
   ModifyTimeDate,                /* time/date of last optical mod*/
   LastAccessTimeDate,           /* time/date of last optical access,
                                          not yet implemented */
   LastSwapTimeDate, /* time/date of last swap not yet implemented */
   LastImportTimeData,             /* time/date volume was imported*/
   LastDatastream,               /* Highest assigned Datastream # */
   IOerrorCount,          /* How many IOerrors not yet implemented */
   AccessCount,             /* How many Access not yet implemented */
   SwapCount;                /* How many swaps not yet implemented */
   BYTE Reserved2[2];                           /* unused two bytes*/
   UINT16 FileSystemDataSize;           /* Size of file system data*/
   BYTE FileSystemData[128];     /* File system specific data area */
}VOLUME_REC;
```

## Volume Record Members

The following section describes some of the members of TagVOLUME_REC in more detail:

`VirtualVid` volume ID of this volume

`VolumeName` The ASCIIZ name of the volume.

`Attributes` The volume attribute double word.

`Privileges` Describes the access rights to the volume. Read-only volumes would be described by this field. This field is not active in the current release of the product.

The privilege field contains the following privileges:

```
#define   READ_VOLUME_DATA    0x00000001
#define   WRITE_VOLUME_DATA   0x00000002
#define   SEARCH_VOL_FILES    0x00000004
#define   DELETE_VOL_FILES    0x00000008
#define   MAKE_VOLUME_DIRS    0x00000010
#define   REMOVE_VOLUME_DIRS  0x00000020
#define   RENAME_VOL_FILES    0x00000040
#define   RENAME_SUBDIRS      0x00000080
#define   RENAME_VOLUME       0x00000100
#define   OVERWRITE_VOL_FILE  0x00000200
#define   CHANGE_FILE_ATTRIB  0x00000400
#define   CHANGE_SUBDIR_ATTR  0x00000800
#define   CHANGE_VOLUME_ATTR  0x00001000
#define   SEARCH_VOLUMES      0x00002000
```

`Time` Encodes the volume mount time.

`Date` Encodes the volume mount date.

`CreateTimeDate` The time this volume was created in seconds past 00:00:00 GMT, January 1, 1970.

`FileSystem-Type` Describes what type of file system is on the volume. The FileSystemType is defined as follows:

Note: All values not defined here are reserved for future use.

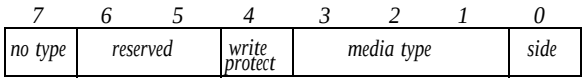
```
typedef INT16 FILE_SYSTEM_TYPE;

#define UNKNOWN_FILE_SYSTEM (FILE_SYSTEM_TYPE)-1

#define PEGASUS_OLD_FILE_SYSTEM
(FILE_SYSTEM_TYPE)0

#define PEGASUS_WORM_FILE_SYSTEM
(FILE_SYSTEM_TYPE)1
```

```
#define ECMA_208_TAPE_FILE_SYSTEM
(FILE_SYSTEM_TYPE)2

#define ISO_9660_FILE_SYSTEM (FILE_SYSTEM_TYPE)3

#define HIGH_SIERRA_FILE_SYSTEM
ISO_9660_FILE_SYSTEM

#define ISO_13490_FILE_SYSTEM (FILE_SYSTEM_TYPE)4

#define PEGASUS_ERASE_FILE_SYSTEM
(FILE_SYSTEM_TYPE)5

#define ISO_13346_FILE_SYSTEM (FILE_SYSTEM_TYPE)6

#define VIEWSTAR_FILE_SYSTEM (FILE_SYSTEM_TYPE)7

#define FILE_SYS_TYPE_COUNT (FILE_SYSTEM_TYPE)8
```

`ReserveSectors` Allows extra sectors to be saved in reserve for performing operations such as delete file after the disk is considered to be full. Any operation can occur if enough space has been reserved for it. Reserved space must be released before it can be used.

`MediaType` A bitwise description of the type of media that contains the volume as being erasable, single-sided, and so on.

MediaType is a single UCHAR that is treated as a group of flags. Interpret the bits as follows:




The *side* bit is set 0 for single-sided, 1 for double-sided. The *media type* bits are set as follows:

0 = Random write-once media
sectors can be written in any order
1 = erasable
2 = CDROM
3 = sequential write-once media
sector can be written sequentially only;
compact disk recordable is an example)

Bit 4 acts as a *write protect* flag. Bits 3, 5, and 6 are reserved. Bit 7 indicates that the media type is unknown.

The following definitions are masks for the disk status information returned in MediaType.

```
#define   MEDIA_MASK       0x06
```

```
#define   WRITE_PROTECT   0x10
#define   WORM_MEDIA      0x00
#define   ERASABLE_MEDIA  0x02
#define   CDROM_MEDIA     0x04
#define   RESERVED_MEDIA  0x06
#define   SINGLE_SIDED    0x00
```

`IsAccessible` A flag that describes whether the volume data is accessible or not. If the disk is off-line, the data is not accessible. If the format is unknown, the data is not accessible. This is a boolean flag where:

1 = accessible
0 = not accessible

`SectorSize` Describes the physical sector size of the volume.

`MaxSector` Contains the logical sector address of the last sector of the volume.

`LocationStatus` Describes whether the volume is on-line (in a drive), off-line (on a shelf), or near-line (in a jukebox). This field can take the following values:

```
#define   OFFLINE           1
#define   ONLINE            2
#define   NEARLINE          3
```

`AliasName` An alternate name for accessing a volume, so volumes with duplicate names can be used in a system.

`DiskName` Contains the low-level disk name which is created at cartridge format time.

`DiskId` Contains a number that uniquely identifies the volume.

`OfflineLocation` A user-entered text string that identifies the off-line location of a volume.

`ExtendedAttribute` A 44-byte extended attribute field that you can use to describe contents of the volume and other pertinent information.

## Record Object Union

The union record is the union of all the possible OFS records. This field allows you to determine what record has been returned to you. The id field defines the record type.

```
typedef struct
{
/* pointer to the file in tree */
drec_ptr recptr;
/* Set to a btree type, TFS_RECORD_TYPE */
UINT16 id;
/* support diff record types */
union
{
   RENAME_REC_STRUC fr; /* file */
   RENAME_REC_STRUC  rr; /* rename file record */
   tfs_subdir_rec    sr; /* subdirectories */
   VOLUME_REC        vr;
   DIR_ENTRY_DESC    dr;
} u; /* end of union */
```

# Return Codes

All functions, unless listed as a boolean, return 0 when successful. If a non-zero value is returned, an error has occurred. For more information, see Appendix A, “Error Codes.”.

# Pathname Rules

InveStore does not use, nor does it understand, drive letters or drive specifiers. Do not include such specifications in any parameter that requires a path. If you do, an error is returned.

# Cache Write Algorithms

InveStore features three Cache Write Algorithms that can be used when writing to and reading from the storage system. A RAM cache layer helps provide for faster write-throughput and less overall disk space usage. Under each write mode, recently active files will remain in the cache buffer and allow files to be recalled instantly.

## Lazy Write

The default and recommended Cache Write setting is Lazy Write. This feature buffers data writes to the optical storage device and flushes files in the background at a later time. This frees an application to continue

with other processes rather than being delayed by the slower speeds of the jukebox and optical drives.

Lazy Write incorporates a background idle time thread for flushing data to optical disk. This helps eliminate the likelihood of performance degradation due to the cache becoming full during heavy operation.

With Lazy Write, InveStore writes data from the RAM cache to optical disk in a contiguous thread to significantly reduces file system overhead by more efficiently using data sectors.

The nature of caching exposes the system to the possibility of data loss if the server should crash prior to the data being committed to optical. InveStore minimizes this risk by periodically committing dirty data to optical automatically.

## Flush on Close

Flush on Close opens and writes files to a RAM cache - which let's InveStore's cache-write algorithms coalesce data to more efficiently fit in given sectors - but performs a full flush of data to optical upon the write close. Once the write finishes, a complete is returned to the application. This method offers a combination of efficient disk usage and the security of writing to disk before freeing the application to move on to other tasks.

## Write Through

Write Through, configures the system to perform write operations directly to the optical disk from file open to file close. This provides the assurance of data being immediately committed to optical – a must for some applications – however results in overall slower throughput and less efficient optical disk space usage

## Optical Storage Considerations

Optical is a significantly slower storage medium than magnetic storage in terms of I/O performance. InveStore caching system is designed to buffer the slower speeds of optical, but as described above, the nature of RAM caching can expose the system to the risk of data loss.

Many applications using optical first capture data directly to large, high performance, magnetic disks, route data through QA and indexing, and at the end of the process archive the data to optical. A prop-

erly designed application will reap the benefits of the InveStore caching algorithm and eliminate risks.

## Lazy Write Application Suggestions

**When writing your application consider using the following scheme:**

1. Data to be stored to optical is captured to magnetic disk as a series of files in a given subdirectory.
2. Once a given number of files have been captured on the front end, the application copies the entire batch of files to optical with Lazy Write enabled.
3. Once the files have been copied the application executes the VIMcommitVolume call.
4. If the call to VIMcommitVolume is successful, the application then knows that all those files have been safely committed. The source file can then be deleted and the process repeated.
5. If the commit call fails, the source files still exists and the copy process can be repeated to the optical system either immediately or once the optical system is functioning properly again if there were specific problems.

In this example, data is maintained on magnetic until it is certain that the data was committed to optical. The system throughput to optical is greatly increased and the file system overhead on optical is significantly decreased. This allows more data to be stored to optical while reducing costs.

**A cautious alternative:**

Cautious applications can consider comparing the source and destination directory entries after the call to VIMcommitVolume to ensure that all the files from the source were on the optical system with the same file size and attributes.

The application can fully control the entire process from filling the source directory, archiving the files to optical, causing the commit to occur, ensuring the commit was successful, and then deleting the source files.

## Identifying Unflushed Volumes

An application can identify which volumes have not been flushed by checking the archive bit for each volume. Each volume which contains information that has not been committed will have the archive bit set in the attribute field. Checking the attributes of the root level sub-directories will tell the application which volumes are dirty.

**Note** Remember that the root level sub-directories are actual optical volume that appear as sub-directories.

Use this as the standard set attribute call to clear the archive bit. This will work across the network.

## When to Use the VIMcommitVolume

An application should allow a substantial amount of data to be written to the optical system before calling VIMcommitVolume. In the case of a batch copy it is recommend that entire batch of files be copied before issuing the commit. If the application is in a continual copy mode then the application can set the commit to occur at given time intervals or after a given amount of data has been copied.

## Other Info

Some additional information about the caching mechanism should be noted. The data of a file is cached to memory. The directory entries upon file close are committed to magnetic disk. Background threads commit both these caches to optical anytime the system is idle. If the system crashes and uncommitted data exists in memory it will be lost but the directory records will exist on the magnetic cache. InveStore will log errors for any file in this cache for which the data does not exist on optical. If the data was committed by a background thread then InveStore will commit this record.

# OPTICAL FILE SYSTEM OVERVIEW

This chapter describes how InveStore manages and stores files on optical media. The topics include:

- System Overview
- Hierarchal Cache Management
- Optical file formats
- Physical file structure
- Logical file structure

## System Overview

InveStore is a complete, high performance optical file server solution for a variety of applications such as imaging and document management. InveStore provides a single, hierarchical file system regardless of the number, or type, of devices that it controls.

### Volume Manager

The Volume Manager has been improved over previous versions. The Volume Interface Module (VIM) now handles any combination of juke-boxes and standalone drives. Different device types, such as a mixture of standalone drives (3.5-inch, 5.25-inch, 12-inch, and 14-inch) and juke-boxes (5.25-inch, 12-inch), can be daisy-chained on the SCSI bus.

## Volume Types

The Volume Manager manages three types of volumes: on-line volumes, near-line volumes, and off-line volumes. On-line volumes are the volumes that are contained within a data transfer unit (disk drive). Near-line volumes are those that are contained within the shelves of a jukebox. Off-line volumes are those that are stored externally to the system (office library for example) and require operator intervention to become on-line. The VIM provides full directory access to all off-line volumes. If a request is received to read data from an off-line volume, the VIM will send a message to any registered console requesting that the given volume be mounted. If confirmation is received that the volume is ready for mounting prior to time-out, the VIM will mount the off-line volume and then service the user request.

## Threads

InveStore is designed for high performance, high volume operations. The Volume Manager and File System kernel are full 32-bit, multi-tasking, and multi-threaded. All threads are created at start-up time. As each request is executed, one of the threads in the pool is used to service the request. If there are more requests than threads in the pool, some requests will be queued at the top level. InveStore has been designed to allow for maximum concurrence, giving the highest throughput possible.

## Optical File Server

The optical file server provides the capability for a separate process to act as a server console. This console provides the ability to mount and unmount volumes, set drive parameters, and report errors. All these activities occur without disrupting the optical file server.

## OHIM

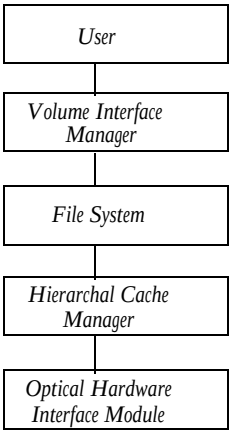
The Optical Hardware Interface Manager (OHIM) has been designed to provide great flexibility in configuring the SCSI bus. The OHIM will support up to four SCSI adapters, allowing the limits of SCSI ID\LUN to be circumvented. The OHIM modules contain their own error logging mechanism, allowing errors for any given device type to be recorded accurately and precise recovery algorithms to be used.

# Hierarchical Cache Management

The Hierarchal Cache Management (HCM) used by InveStore is an advanced caching approach that best fulfills the needs of a varied user group. Unlike Hierarchal Storage Management (HSM), HCM operates below the volume manager and file system, where it can optimize file access. HCM significantly improves the performance of an optical file system in two ways

- Caches large amounts of data to minimize access to the optical system
- Manages the optical disk storage in a way that minimizes head movement and disk swapping

The figure below shows where HCM fits in the file system hierarchy. Because HCM is close to the optical hardware, the volume manager and file system can work in concatenation with HCM to provide the best possible performance.




**HCM functional diagram**

## HCM Operation

The InveStore implements two levels of HCM, accessed in order of speed:

- RAM cache
- Optical

RAM cache yields the fastest access times, but its small size makes caching to DASD a more important optimization. A future release of the InveStore will incorporate complete hard disk caching.

HCM works as follows. The entire directory structure is cached to DASD in an optimized B+tree file(s). If requested, file reads or writes are also cached. There is no notion of high and low water marks because an asynchronous, low priority thread cycles through the cache dumping dirty buffers during idle time. When the cache is full, HCM flushes all of the buffers for the current volume, then for the least recently used volume. This is a volume-centric approach that has the following advantages:

- Writes to a single platter, the one mounted if possible, to avoid platter swaps
- Writes a group of buffers in one operation, generally in ascending sectors to optimize performance

Avoiding platter swaps is an important optimization because a single swap can take 30 to 40 seconds to complete.

### HCM Implementation

The current InveStore implementation of HCM supports the following features:

- WORM and rewritable media
- High Performance Worm Optical File System
- RAM HCM layer on reads and writes

## File Formats

InveStore supports the following file formats:

- HPWOFS (High Performance Worm Optical File System) Revision 1, a write once, read multiple format.

- HPWOFS Revision 2 a new enhanced format for worm optical media the supports larger file sizes, longer file names, multiple data streams, and a larger file attributes space.
- Default format for media type, for example ISO 9660 level 1 or Joliet for CD-based media.

## HPWOFS Rev 1

The standard HPWOFS Rev 1 include the following features:

- 4 G file size
- 19-character file names
- A single data stream
- DOS-based TIMEDATE

### Handling HPWOFS Rev 1 Data

Continue to write HPWOFS Rev 1 data in the Rev 1 format until you reformat the platter. (Only applicable to MO disks). InveStore maintains the old format on these disks. Format new disks in the new Rev 2 format.

## HPWOFS Rev 2

The main features of the HPWOFS Rev 2 are:

- Larger files: 10 terabytes
- Longer file name: 256-bytes (v3.10.00 and higher)
- Multiple data stream structures
- Larger file attributes space
- Backward compatibility

Previous releases of Pegasus cannot read data written under HPWOFS Rev 2.

**Note** HPWOFS supports single surface file systems only (no platter spanning).

### WORM

WORM optical media are by definition non-erasable. Once written, space on the optical disks cannot be reclaimed. This physical difference between WORM optical media and magnetic media has important implications for the physical organization of the optical disk. It affects how the disk is divided into physical areas and how data is added to these areas; it gives new meaning to the DISK_FULL condition.

The areas of a WORM optical disk are physically defined; their addresses are defined in terms of absolute locations on the optical disk.

The physical layout of WORM optical media is a little different from magnetic media. Magnetic media is a disk composed of physical cylinders; a cylinder is composed of physical sectors; a sector is composed of physical blocks; and a block is composed of physical bytes.

Optical media does not use cylinders. The disk is composed of sectors. Sectors and blocks are the same thing. The number of bytes in a sector is device-dependent. To find out how many bytes are in a sector on your WORM optical drive system, see the documentation for that system.

### OSTA-UDF/ECMA 167 Compliant

InveStore supports the volume recognition sequence and the volume structure of the standard through part three. For more information, see the following web sites:

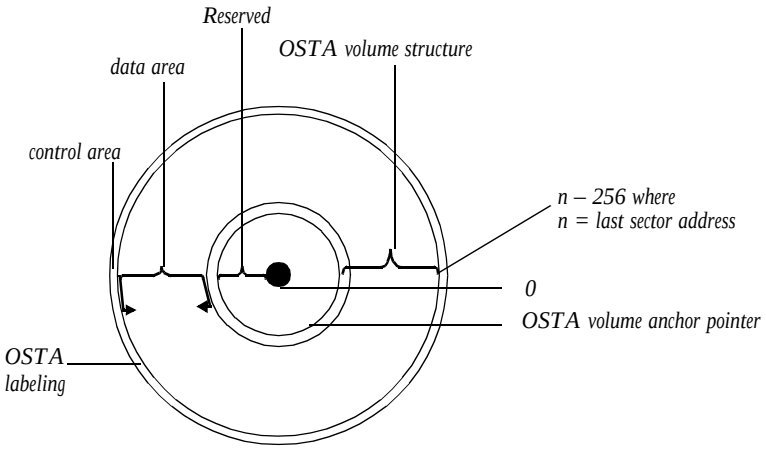
- http://www.osta.org
- http://www.ecma.ch

## Physical File Structure

In accordance with the Small Computer Systems Interface (SCSI) standard, the sectors on a WORM optical disk are numbered sequentially. The first sector, located on the inner edge of the disk, is sector 0. The total number of sectors on the disk is also device dependent. The last sector is located on the outer edge of the disk; the number of the last sector can be determined as follows:

Last sector number = (Number of sectors on disk – 1)

A WORM disk has a Control Area and a Data Area contained within the partition defined by the OSTA and ECMA volume structures. The following diagram shows the architecture of the physical file system of HPWOFS optical disks:




**Physical Architecture of an Optical Disk**

The first 256 K of the optical disk is reserved. InveStore puts the volume recognition sequence at the defined area of 32 K. At 256 K and n–256 K is the OSTA anchor volume pointer. This pointer identifies the OSTA volume descriptor sequence and the area containing all OSTA volume labeling information, including a definition of the partition for InveStore. The file system within the partition is proprietary.

## Control Area

The Control Area contains the disk's directory information. It starts at the last sector of the partition—the outermost sector on the partition—and grows inward toward the center of the disk. Every time information is written to this area, the disk is checked for the DISK_FULL condition.

The last sector, minus 256, of the disk contains the OSTA volume anchor pointer.

## Data Area

The Data Area contains the files that are stored on the disk. Use the OSTA structures to determine where the partition starts. From the starting point, the data area grows outward toward the center of the disk.

**Note** All structures (minus the anchor volume pointer) can be placed anywhere and have no set location.

Within the Data Area, the same file can be stored across many sectors, and the sectors within which a particular file is stored can be separated by sectors containing other files. Whenever a file is written to the disk, it is checked for the DISK_FULL condition.

### The DISK_FULL Condition

The DISK_FULL condition occurs when there are less than 2M bytes of room left on the disk. The allocation of the Control and Data Areas puts the last 2M bytes somewhere in the middle of the disk.

## Logical File Structure

The Logical File Structure of InveStore is composed of a set of software control structures (control records) and the directory structure into which these records are organized.

## File System Operations

Standard file operations, including file open, file create, directory create, file read and file write, appear to the programmer to function in InveStore exactly as they do in the NT file system.

With CD/DVD-ROM media, standard file operations including file open and file read appear to the programmer to function in InveStore as they do in the MS-DOS file system. CD-ROM and DVD-ROM based systems differ from WORM and rewritable system in terms of what operations are allowed. CD/DVD-ROM media is read-only, so nothing can be written to disk.

The InveStore Volume Interface Module (VIM) treats all the mounted optical volumes as if they comprise a single file system. It reserves the root directory of InveStore for a special purpose.

The volume name of each mounted volume is used as a subdirectory name in the root, and there is a subdirectory name for every mounted volume. This means that no other subdirectories or files can be placed in the root. Such inappropriate files would be treated as if they were volume names, which can lead to fatal InveStore sys-

tem errors. The only valid operations on the root node are searches and directory changes.

## File Accessibility

With WORM and rewritable media, any file residing on the optical disk can be opened once (have one file handle opened) for READ/WRITE access. This same file can be opened up to the maximum number of file handles for READ ONLY access. In other words, WRITE access is exclusive, and READ access is non-exclusive.

With CD/DVD-ROM media, any file residing on disk can be opened up to the maximum number of file handles for READ ONLY access.

# SYSTEM SOFTWARE

The InveStore development system consists of several modules that you must understand to write an application interface. Modularity makes the code easier to modify, maintain and extend. It makes possible an optical file system that is hardware-independent in the sense that it can support devices manufactured by any vendor with the addition of a single module. The only requirement is that the drive system interface conform to SCSI standards.

This chapter discusses the following topics in detail:

- Structure and function of the primary system modules
- The flow of a file system request through system modules
- User-configured system parameters
- Initialization of InveStore

# InveStore Kernel

InveStore employs a pool of threads to process requests and improve performance under Windows NT/2000. Thread creation and destruction is more efficient using the pool. Configuration parameters for the number of threads and number of queued I/O requests are located in the following registry locations:

Parameters related to file system use are located in:
HKEY_LOCAL_MACHINE/SOFTWARE/Pegasus/InveStore/Pdtcp

Keys

**NumberOfThreads (REG_DWORD) -** Number of threads in the worker thread pool

**NumberOfOverlaps (REG_DWORD) -** Number of I/O requests pre-queued for the file system request processing.

Parameters related to the VIM API processing (including GUI):
HKEY_LOCAL_MACHINE/SOFTWARE/Pegasus/InveStore/Kernel

Keys

**NumberOfThreads (REG_DWORD) -** Number of threads in the worker thread pool per requester process. If you run the GUI and some other application making VIM calls, each will get its own set of threads.

**NumberOfOverlaps (REG_DWORD) -** Number of I/O requests pre-queued for the VIM API request processing per requester process.

To access cache files, the kernel draws upon a pool of file handles, which is of the size defined by the user in the CACHE HANDLES parameter in the OPTICAL.CNF file.

The InveStore kernel is designed to be both hardware- and operating system-independent. Functionality required to deal with devices and their specific physical and logical characteristics are isolated in the Optical

Hardware Interface Module (OHIM). Functionality that is specific to the host operating system is isolated in the installable file system (IFS) layer.

The InveStore kernel is responsible for implementing the hierarchical tree that forms the structure of the file directory; it builds this tree in its internal cache. The kernel provides a full set of standard file facilities. It provides the ability to create, delete, update, append, rename, modify, and truncate files and to create subdirectories. The theoretical maximum file size is calculated as follows:

*max file size = total bytes on disk – (system reserve area (2 MB) + user reserved area + size of 2 control sectors)*

where,
user reserved area varies, default = zero.

Since in this case there would be only one file on the disk, the Control Area would consist of two sectors.

The InveStore kernel is composed of the following seven major modules:

- Volume Interface Manager
- Logical File Server
- Physical Format Manager
- Hierarchical Cache Manager
- Physical File Server
- Directory Cache Manager
- Memory Manager

## Volume Interface Manager

The Volume Interface Module (VIM) makes the optical directory appear as a single logical structure. To accomplish this, the VIM creates a subdirectory in the InveStore root as each volume is mounted. The VIM is the only system software that deletes these volume subdirectories, an event associated with the logical unmount operation. When queried about mounted volumes, the VIM displays information for all known volumes, some of which may not be physically mounted nor contained within a jukebox.

The VIM parses the path and determines on which volume the target file is located. Once it has gathered together all necessary volume

information on the target file, including the size of the file and its location in the volume, it passes this information along to the InveStore kernel.

Now that the target file has been found and sized, all that remains for the InveStore kernel to do is to assign the necessary tasks to its task servers or managers, as appropriate. The kernel needs to set up the file transfer from disk into memory, so it allocates buffer space and does other housekeeping chores, such as creating a File Control Block for this particular transaction. Once the kernel has completed these tasks, it calls the OHIM to access the hardware.

## Logical File Server

The Logical File Server functions as the primary interface between the kernel and the Volume Manager. The Logical File Server acts as a "front end" to the kernel by processing file system requests. These requests are submitted to this server in one of two ways. They may come in from the transparent OS call or directly through the VIM. The Logical File Server parses request packets and dispatches the other kernel-based tasks necessary to service the request.

## Physical Format Manager

The Physical Format Manager understands and maintains the physical file system structure of the underlying media. There is a PFM for each file format supported by InveStore, including HPWOFS Rev 1 and HPWOFS Rev 2, ISO-9660 and Joliet.

## Hierarchical Cache Management

The Hierarchical Cache Management (HCM) used by InveStore is an advanced caching approach that significantly improves the performance of an optical file system. For more information, see "Hierarchical Cache Management" on page 43.

## BTREE and Directory Cache Manager

The BTREE manager maintains the system's directory caches. The Directory Cache Manager loads the initial directory into the cache buffers and swaps new directory information between the buffers and magnetic media on an as-needed basis. For more information, see "Hierarchical Cache Management" on page 43.

### Physical File Server

The Physical File Server handles input from and output to the optical drive system. It is called by the Logical File Server to oversee the transmission of any data necessary to service a file system request. The Physical File Server is responsible for the data blocking and deblocking necessary to translate data from physical sectors to logical buffers. The primary control structure used by this file server to accomplish its tasks is the File Control Block (FCB).

In addition to its blocking/de-blocking functions, the Physical File Server also handles bad sector mapping and disk errors.

### Memory Manager

The Memory Manager is responsible for allocating and maintaining the cache and data buffer pools in accordance with the parameters specified by the user.

## User-Configured System Parameters

This section briefly describes system parameters for tuning the performance of InveStore. For more information on performance, see the *InveStore User Guide*, Chapter 5, "Optimizing System Performance."

Memory usage, caching options — to mention a few — can be configured easily for your application. The manner in which drives are allocated to volumes can also be modified.

Both the minimum time and maximum time a volume can remain in a drive can also be set, giving priority to read requests over write requests and vice-versa. Additionally, you can tell a particular drive only to service read or write requests or take a problematic drive offline, via software, without physically requiring that a disk be removed.

The parameters in OPTICAL.CNF fall into four categories:

- General
- Mount
- Cache
- Disk Swapping

## General Parameters

**HANDLES** The number of files that can be open at one time by all users of the optical drives.

Valid Range: 1 – 1024
Default: 64

**PARTITION** The amount of free disk space (in megabytes) required for running InveStore.

Valid Range: 5 – 2000
Default: 5

**REPLACE EMPTY FILES** The parameter that controls whether or not zero-length files stored on optical media are replaced when you attempt to create a file by the same name. When REF=Yes, InveStore treats a zero-length file as if it doesn't exist, and creates a new file of the same name as the zero-length file.

Default: NO

**RETRY OPTICAL ERRORS** The parameter that determines whether or not certain errors returned from the optical device drivers (OHIM) are retried (during write requests).

Default: YES

**DELETE FILES** Specifies whether or not files can be deleted. The default of YES allows files to be deleted. Setting this to NO will cause all Delete Files requests to be rejected. This should be used if you want to prevent users from accidentally deleting files from optical disk.

Default: YES

## Cache Parameters

**DEFAULT CACHE WRITE ALGORITHM** A parameter set by the administrator to define the default cache write algorithm, so caching can occur if the Open/Create command does not specify caching. Possible values are LAZY_WRITE—data is stored in cache and flushed to optical at a later time; FLUSH_ON_CLOSE— data is stored in cache but flushed to optical immediately when the application issues a file close; WRITE_THROUGH—data is

written immediately to optical from file open to file close.

**DEFAULT MINIMUM CACHE PAGE** — Defines a default minimum cache buffer page size for the system, which is used as the default during mounts if not overridden by the user. Minimum value: 4 K; Maximum value: 64; Default: 4

**DEFAULT MAXIMUM CACHE PAGE** — A read-only value for the default maximum cache page size, in kilobytes. The minimum value is 4, the maximum 64, and the default is 64. The Minimum Cache Page value should be less than or equal to the Maximum Cache Page value.

**MAXIMUM RAM CACHE BUFFERS** — Defines the maximum RAM cache that can be allocated, *not* how much memory might directly be allocated; dictates the size of internal structures that create allocation structures. Actual data buffers are allocated as needed.

**MIN/MAX CACHE PAGE** — Unused. Reserved for DASD cache feature.

**CACHE MEMORY SIZE** — Defines the amount of memory in Kb reserved by the system for cache descriptors. This does not include data buffers, which are allocated as needed,

but limited by the amount of descriptor memory allocated. Default is 16Kb.

**LAZY WRITE** Output data is stored in cache and written to optical at some indefinite later time.

**FLUSH ON CLOSE** Output data is written to RAM but flushed to disk immediately when the application issues a file close.

**WRITE THROUGH** Output data is written to optical media at the time it is presented to the caching subsystem. It may remain in the cache for faster access reading.

**MOUNT VALIDATION** Not implemented. Reserved for future use.

**CACHE COMMIT THRESHOLD** The time interval after which data written to the cache is eligible to be committed to optical media automatically. The default is two minutes.

**CACHE FILE POSITION THRESHOLD** The relative position within a file beyond which data will not be retained in the cache. This prevents a large sequential file transfer from forcing data out of the cache and improves caching performance because large, sequentially accessed, files cannot be cached effectively anyway. The default is one Megabyte.

**CACHE BLOCK SIZE THRESHOLD** The size of a data buffer at or above which the CACHE FILE POSITION THRESHOLD will apply for reads. The default is zero; the maximum is 64 Kb. Setting this parameter nonzero can permit direct access to small blocks within a large file to be cached, while excluding large sequential data transfers from being cached.

## Mount Parameters

**PREMOUNT ALL DISKS** Checks the cache files for every volume within each jukebox during initialization (jukebox start-up sequence). If Premount All Disks is unchecked (no pre-

mounting), InveStore does not validate the cache files for each disk if the following conditions exist:

1) the kernel finds no disks in a slot it thought was empty

2) all the slots that it thought were occupied are still occupied.

If both conditions are met, the cache files are not validated until the first time the disk is accessed. If the cache file is corrupt, opening the first file on the disk can take a long time.

Default: YES

**MOUNT VALIDATION** Causes the system to validate each file while mounting if set to TRUE. Normally, set this option to FALSE. Set to TRUE only when you have a damaged disk, because validation seriously degrades performance during mounting.

Default: NO

## Disk Swapping Parameters

**Note** The value 65535 for any time out disables the time out (infinite time).

**REQUEST MOUNT TIME** The amount of time InveStore waits for you to mount a disk after requesting an offline volume. The request to search a directory is not affected by this parameter. Set this parameter to a value that gives you enough time to find and mount disks. When the value is too high, applications appear to be hung when you don't mount the disk.

**Note** InveStore will suspend *any* request for an offline volume for as long as specified by this parameter or until the disk is mounted.

When you set this parameter to 0, all requests to an offline volume, other than searching a directory, fail and an access denied error message is displayed. When set to greater than 0, InveStore waits the specified time for

you to mount the requested volume, and if it is not, an error message is displayed.

Minimum value: 30 seconds
Maximum value: 65535 (infinite wait)
Default: 0 seconds (request fails immediately)

**REQUEST WAIT TIME** The maximum amount of time in seconds that a user request waits for a drive to become available, before InveStore returns a request failure error to the user. The user request might be a request to read or write or some other form of physical disk access such as "make directory" to / from optical disk. Any request that exceeds the request wait time is terminated with an error.

Minimum value: 30 seconds
Maximum value: 65535 (infinite wait)
Default: 300 seconds

**PRIORITY WAIT TIME** The amount of time in seconds that a request waits before it is moved to the high priority queue.

Minimum value: 15 seconds
Maximum value: 65535 (never move to higher priority)
Default: 30 seconds

**LOCK TIME** The minimum (or maximum) amount of time in seconds a volume remains in the drive before it can be swapped out by another request.

Minimum value: 15 seconds; Maximum value: 65535 (continuous lock during requests); Default: 60 seconds

**HOLD TIME** The minimum amount of time that a disk will remain in a drive after the last request was received for that disk.Valid Range: 1 second – 30 seconds
Default: 2 seconds. The following example is for a hold

time of 2:

1) User A requests data from volume A.

2) Volume A is loaded into the drive (this example uses a single drive system).

3) User receives Data and the lock on the drive is released (drive is now available. There are no other pending requests for volume A).

4) User B requests data from volume B.

5) User B's request will not be honored until Hold Time seconds have passed since the last request for volume A was received. This will allow another request for volume A to be processed prior to processing the request for volume B. If the Hold Time seconds pass and there are no requests for volume A, volume A will be swapped out and the request for volume B will be processed.

The intent of Hold Time is to reduce the total number of disk exchanges and improve system throughput.

**SYSTEM BIAS**

The drive bias being used by default. The System BIAS is the factor that determines which type of I/O operation the system favors as the default. System BIAS defines which requests (read or write) are processed or if the default behavior is read only or write only. This parameter might be used, for example, in a 4-drive jukebox when you want only one drive available for writes, so you set three drives to read only. With the BIAS, the system administrator can optimize the sharing of the drive resources. System BIAS an be overridden by setting it from the graphical user interface (Library/Drive Settings option in the Edit menu).

Parameters: None, ReadOverWrite, WriteOverRead, ReadOnly, WriteOnly. Default: None

**OFFLINE DRIVES**

This parameter defines the off-line drives in the system. If a drive is set offline, it will not be accessible by InveStore. For this parameter to exist in the OPTICAL.CNF file, you must define the drive as off-line. Use the

Library/Drive Settings option from the Edit Menu to set this parameter.

Example: OFFLINE DRIVES = 1

Library drives are numbers starting at 0.

**Warning**: If you add or remove hardware or change a SCSI ID, ensure that this parameter is correct for the new configuration.

**DRIVE BIASES** This parameter defines the biases for each drive. When set, it overrides the System Bias. This parameter will exist in the OPTICAL.CNF file only if the drive bias is set to something other than System Bias. Use the Library/Drive Settings option from the Edit Menu to set this parameter.

Example: DRIVE BIASES = 0:READ
where 0 represents the drive number

Possible values are

NONE
READONLY
READ
WRITE
WRITEONLY

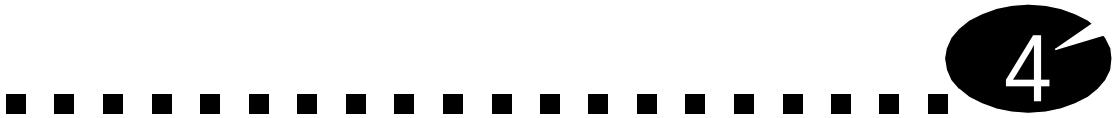
# InveStore Initialization

The InveStore kernel begins by determining whether an OPTICAL.CNF file exists. If it does, InveStore reads the file. Then, InveStore configures the system by allocating buffers and setting initial values in accordance with the user-configurable parameters. (If the OPTICAL.CNF file does not exist, InveStore uses the default configuration values.)

The behavior of the InveStore kernel components at InveStore initialization time is as follows:

- The Logical File Server initializes system variables.
- The Physical File Server initializes file handle tables.
- The Magnetic Support Manager performs an existence check on the system hard drive.
- The Memory Manager allocates the required system buffers.
- The Cache Manager initializes the cache buffers.
- The Physical Control Manager sets up additional physical interface tables.

The OHIM checks for the existence of an optical drive system and queries the existing system to set up its internal SCSI-drive tables.



# API REFERENCE GUIDE

This chapter describes the Volume Interface Module (VIM) calls, including the new calls, VIMopenCreate and VIMcommitVolume.

**Note** The Pegasus VIM interface is subject to additions and modifications without notice. New calls are added as the need arises.

Always obtain the latest header files and full source code examples from the web site http://www.pegasus-ofs.com. It contains the latest call enhancements and updates to documentation.

## VIM Interface Calls

This section describes the VIM interface calls in alphabetical order.

### VIM_chdir

This function call is used to change the current directory.

#### Syntax

```
int ENTRY VIM_chdir (pTEXT path,
   CurDirInfo PTR cur_dir_input,
   CurDirInfo PTR cur_dir_output)
```

#### Input

`path` The desired new current directory.

`cur_dir_input` Provides the kernel specific information for the current directory. If `cur_dir_input` is valid, the directory will be changed relative to the current directory.

### Output

If the call is successful, the new current directory information is stored in `cur_dir_output`. Returns 0 if successful; otherwise, an error is returned. The `cur_dir_output` can then be used to supply current directory information to other calls.

### Example

```
APIRET rc;
CurDirInfo LocalCurDir;
CurDirInfo NewCurDir;
CHAR szPathName[256];
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
   LocalCurDir.volume_id = NO_VOLUME;
   strcpy (szPathName, \\MYVOLUME\\MYSUB1\\MYSUB2);
rc = (APIRET) VIM_chdir (szPathName,
   &LocalCurDir, &NewCurDir);
```

## VIM_check_eof

This call checks the specified handle to determine if the file is at end-of-file condition.

### Syntax

```
int ENTRY VIM_check_eof (int file_handle)
```

### Input

`file_handle` Handle that was returned from a prior successful open\create call.

### Output

Returns 0 if not at end of file, EOF_ERR if at end of file, or possibly invalid file handle.

## VIM_cleanup_process

This call is used to clean up all resources associated with a process when that process terminates.

### Syntax

```
int ENTRY VIM_cleanup_process (void)
```

### Output

Return code, 0 = success.

## VIM_close

This call closes the file associated with the supplied file handle.

### Syntax

```
int ENTRY VIM_close (int file_handle)
```

### Input

`file_handle` The handle that was returned by a previous successful open\create call.

### Output

Returns 0 for success; otherwise, a host of errors can be returned including write errors, invalid file handle, magnetic disk I/O failure (spool cache).

### Post-Conditions

If the file was opened for writes, all data is flushed to optical and the final directory record is recorded.

## VIMcommitVolume

The GUI uses this call for the Commit Volume option on the File menu. VIMcommitVolume commits all cached data for a given volume to optical disk. The application can take advantage of lazy write algorithms and then commit the data when necessary.

The call allows an application to write a number of files to a given optical volume, using the lazy write capabilities, which optimizes performance. The application calls VIMcommitVolume periodically to ensure that all the files were written successfully.

The application maintains a transaction list of file names and sizes that were sent to the optical disk using lazy write. If the call to VIMcommitVolume fails, the applications searches the optical disk for files names

on the transaction list. When it finds a file, it compares the size of the file on the disk against the file in the list, and knows which files were successfully committed.

### Syntax

```
INT VIMcommitVolume (pText Volumename);
```

### Input

`Volumename` The name of the volume to be flushed. This value is obtained from the Volume Record. For more information, see “Volume Record” on page 31. You can obtain the volume record by using VIM_search or VIM_get_object_info.

Note: The archive bit is set if the volume is dirty.

### Output

Returns 0 if successful; otherwise, an error code is returned.

### Example

```
APIRET rc;
CHAR szVolName[256];
strcpy (szVolName, "\\MYVOLUME");
rc = (APIRET)VIMcommitVolume( szVolName );
```

## VIM_create

This call is used to create a new file or to overwrite an existing file. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_create (pTEXT file_name,
   CurDirInfo PTR cur_directory,
   pNINT file_handle, ATTRIBUTE32 attributes,
   int mode)
```

### Input

`file_name` The name of the file to be created (with or without path information).

`cur_directory` Supplies the kernel-specific information about the current directory.

`attributes` Contains the desired attributes for the new file. For more information, see “Common Fields” on page 28.

`mode` Describes the action to be taken:

0 = overwrite if file exists
1 = fail if file exists

### Output

If successful, the handle number is returned in `file_handle`.

## VIM_delete

This call deletes the file specified by file name (with or without path information). This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_delete (pTEXT file_name,
   CurDirInfo PTR cur_directory)
```

### Input

`file_name` Describes the file to be deleted in ASCIIZ format. `file_name` can contain path information. `file_name` cannot contain wildcard characters. Only one file at a time can be deleted.

`cur_directory` Supplies the kernel with information about the current directory.

### Example

```
APIRET rc;
CHAR szFileName[256];
CurDirInfo LocalCurDir;

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szFileName, "\\MYVOLUME
   \\MYSUB1\\FILE.IMG");
rc = NO_ERROR;
rc = (APIRET) VIM_delete (szFileName, &LocalCurDir);
```

# VIM_disk_params

This call supplies disk information for a given volume.

## Syntax

```
int ENTRY VIM_disk_params(pTEXT volume_name,
   CurDirInfo PTR Cur_directory,
   pUSHORT sector_size,
   pULONG total_sectors,
   pULONG free_sectors)
```

## Input

`volume_name` The optional name of volume for which information is desired. If no name is supplied, this variable should be NULL. If the volume does not exist, an error is returned. It is recommended that you always use the volume name.

`cur_directory` Contains current directory information. You must either supply the volume name or a valid current directory structure. An error is returned if both are missing.

## Output

If successful, the sector size, total number of sectors, and number of free sectors are returned.

`sector_size` Contains the physical sector size of the volume.

`total_sectors` Contains the total number of sectors for this volume.

`free_sectors` Contains the number of available sectors on the volume.

## Pre-conditions

If a volume name is not supplied, the caller must have a current directory. This is the only way to determine for which volume this call is intended. If no volume name is supplied and there is no current directory besides the root, the function returns an error 0x00E4 (No volume ID).

## Example

```
APIRET rc;
CHAR szVolName[256];
USHORT usSectorSize;
```

```
ULONG ulSectorTotal, ulSectorFree;
CurDirInfo LocalCurDir;

strcpy (szVolName, "\\MYVOLUME");

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
rc = (APIRET) VIM_disk_params (szVolName,
   &LocalCurDir, &usSectorSize, &ulSectorTotal,
   &ulSectorFree);
```

## VIM_flush_buffers

This call is used to flush and sync the DASD cache and optical buffers.

### Syntax

```
int ENTRY VIM_flush_buffers (void)
```

### Output

Returns 0 if successful; otherwise, an error code is returned.

## VIM_format

This call is used to format a disk cartridge. This function is not applicable to the CD/DVD-ROM software.

This call works for standalone drives, rapid changers, and jukeboxes. Trapped volumes are not supported for standalone drives; therefore, do not pass a trapped volume name, !INI (blank), !PDT (unrecognizable file system), !FMT (failed during format), and !ERR (failed during mount), when calling this function to format a disk in a standalone drive. To use this call with a standalone drive, place the disk in the drive, and then call this routine. Pass null in the second parameter (trapped volume name).

This call supports formatting virgin disks that are manually inserted or mounted into the jukebox (preloaded cartridges !INIxxx volumes). This call effectively handles only single-access media (LMS 4100 or 6100) because it only formats one side of a cartridge. This is not to say that the call cannot be used for double-sided media. You must repeat the call if you want to format the other side of the media. The main purpose of this call, however, is to format volumes within

a jukebox or to process single-access media. This call does not import media from the mail slot; Use VIM_format_media.

**Note** This call has been changed in this release to allow formatted volumes already existing within a drive or jukebox to be reformatted.

To reformat a volume, pass the existing name of the volume in TrappedVol. Pass the new name of the volume in VolName.

### Syntax

```
int ENTRY VIM_format (DID DeviceID,
   pCHAR TrappedVol,
   pCHAR VolName,
   int MountAfterFormat)
```

### Input

`DeviceID` A logical number from 0 to *n*-1, where *n* is the number of drives being managed by the OHIM.

`VolName` The desired volume name of the cartridge for side 1, or, in the case of single-access media, the volume name for the entire cartridge. This volume name appears as a subdirectory of the root of the file system. This name must follow the naming convention for subdirectories.

This call does not provide for a user input disk name; it simply uses the volume name as the input for the low-level disk name.

`TrappedVol` The name of the unformatted cartridge within the jukebox or Rapid Changer. The name of a trapped volume is automatically generated by InveStore during the mount procedure or during the initialization procedure if the virgin disks were hand-loaded into the jukebox when the system was turned off. The name of a trapped volume has one of the following prefixes:

!INI*xxxx* — defines a format that is not a PDT format. InveStore numbers blank files sequentially and adds a prefix of !INI. !INI is suffixed with the 4 hexadecimal digits of the OHIM Slot ID (*xxxx*) of the disk (two per double-sided volumes).

!PDT*xxxx* — means the disk contains an unidentified file system. !PDT is suffixed with the 4 hexadecimal dig-

its of the OHIM Slot ID (*xxxx*) of the disk (two per double-sided volumes).

!FMT*xxxx.yyy* — means that the disk failed to successfully complete a FORMAT operation (either PreLoad Initialize or Initialize/Reinitialize). !FMT is suffixed with the 4 hexadecimal digits of the OHIM Slot ID (*xxxx*) of the disk (two per double-sided volumes) followed by the hexadecimal error code returned by the format routine (*yyyy*).

!ERR*xxxx.yyy* — means that the disk failed to successfully mount. !ERR is suffixed with the 4 hexadecimal digits of the OHIM Slot ID (*xxxx*) of the disk (two per double-sided volumes) followed by the hexadecimal error code returned by the mount routine.

!PFMxxxx.yyy — means that we recognize the format, but could not successfully load the DLL that can process this format. !PFM is suffixed with the 4 hexadecimal digits of the OHIM Slot ID (*xxxx*) of the disk (two per double-sided volumes) followed by the hexadecimal error code returned by the mount routine.

Trapped volumes are not supported for standalone drives. Set this text to NULL if you are not processing a trapped volume.

`MountAfterFormat` A flag that is set to 1 if you want the disk to be mounted after it is formatted. Set this flag to 0 if you want the disk to be ejected after the format. Generally, you will always set this parameter to 1 to mount the disk after formatting. Usually you use this call to format and mount numerous preloaded blank media.

**Note** It is illegal to attempt to eject after format any disk that is double-sided, because the disk is mounted and available to users.

You cannot set the MountAfterFormat to 0 if the media is double-sided and inside a jukebox or if the media is standalone.

## Output

Return code, 0 = success

### Example

This example shows how to format a blank disk from slot 0 in Library 0, with volume name/disk name "0008". To mount the volume after format using the WOFS REV2 format, see “VIM_format_media” on page 74.

```
APIRET rc;
rc = (APIRET) VIM_format (0, "!INI0000", "0008", 1,
1);
```

## VIM_format_media

The GUI uses this call to implement the Initialize function. This call is used to format a disk cartridge. It works for standalone drives, rapid changers, and jukeboxes. This function is not applicable to the CD/DVD-ROM software.

In the case of a jukebox, the cartridge is expected to be imported from the mail slot. This call does not format virgin disks that are hand stuffed into the jukebox (pre-loaded cartridges); use VIM_format for this purpose. VIM_format_media allows formatted volumes already existing within a drive to be reformatted. It allows unmounted InveStore volumes to be reformatted. It will not allow reformatting of mounted volumes. This call handles both single-access media (LMS 4100) and the normal double-sided media. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_format_media (pTEXT VolName1,
   pTEXT DiskName1,
   pTEXT VolName2,
   pTEXT DiskName2,
   DID DeviceId,
   int Reformat,
   int MountAfterFormat,
   int FileSystemType)
```

### Input

`VolName1` The desired volume name of the cartridge for side 1, or, in the case of single-access media, the volume name for the entire cartridge. This volume name appears as a subdirectory of the root of the file system. This name must follow the naming convention for subdirectories.

`DiskName1` The desired low-level disk name of the cartridge for side 1, or in the case of single-access media, the disk name for the entire cartridge. This name is not used to access the cartridge. It is a means of uniquely identifying the cartridge and its contents.

This name can be up to 20 characters long. Embedded blanks are allowed.

`VolName2` Same as `VolName1`, but meant for side b of a double-sided media. Set this pointer to NULL for single-access media.

`DiskName2` Same `DiskName1`, but meant for side b of double-sided media. Set this pointer to NULL for single-access media.

`DeviceID` A logical number from 0 to $n - 1$, where $n$ is the number of drives being managed by the OHIM.

`Reformat` A flag that is set to 1 if you want to reformat an existing volume or one that has already been formatted. Set this flag to 0 for unformatted media.

`MountAfter-Format` A flag that is set to 1 if you want the disk to be mounted after it is formatted. Set this flag to 0 if you want the disk to be ejected after the format.

`FileSystem-Type` Defaults to the Pegasus WORM file system, REV2. The REV1 format is not supported.

## Output

Return code, 0 = success

## Example

```
APIRET rc;
   UCHAR szVolName1[50];
   UCHAR szDiskName1[50];
   UCHAR szVolName2[50];
   UCHAR szDiskName2[50];
DID Drive;
   int Reformat;
   int MountAfterFormat;

   memset(szVolName1, 0x00, sizeof(szVolName1));
   gets(szVolName1);
   strcpy(szDiskName1, szVolName1);
```

```
memset(szVolName2, 0x00, sizeof(szVolName2));
gets(szVolName2);
strcpy(szDiskName2, szVolName2);
Drive = (DID) 0;
Reformat = REFORMAT_FALSE;
MountAfterFormat = (int) FORMAT_MOUNT;
rc = (APIRET) VIM_format_media(szVolName1,
    szDiskName1, szVolName2, szDiskName2,
    Drive, Reformat, MountAfterFormat, 1);
```

## VIM_fseek

The call is used to change the position of the file pointer. This call works just like the CLIB function lseek.

### Syntax

```
int ENTRY VIM_fseek (int file_handle,
   long desired_offset,
   pULONG absolute_offset,
   int mode)
```

### Input

`file_handle` The desired file for the operation. `file_handle` was returned from a prior successful open\create.

`desired_offset` The required new file pointer position.

`mode` Determines how the function works.

mode 0 = move file pointer relative to the beginning of the file.

mode 1 = move file pointer relative to the current file pointer position.

mode 2 = move file pointer relative to the end of the file. Setting desired offset to zero has the effect of returning the file size.

### Output

If successful, the actual file position is returned in `absolute_offset`.

## VIM_get_drive_bias

The GUI uses this call to implement getting drive bias in the Library Status item on the Edit menu. This call is used to determine the bias of the requested drive.

With the CD/DVD-ROM software, this function is always set to None.

### Syntax

```
int ENTRY VIM_get_drive_bias(DID DeviceId, int*bias)
```

### Input

`DeviceId` A logical number from 0 to *n*-1, where *n* is the number of drives being managed by the OHIM.

### Output

`bias` Contains the desired bias. The values of bias are defined as follows:

bias = 0 means the drive has no bias. This indicates that all I/O requests are treated equally and are processed in an FIFO manner.

bias = 1 means the drive has Read Over Write bias. This indicates that read requests have priority over write requests and are processed first.

bias = 2 means the drive has Write Over Read bias. This indicates that write requests have priority over read requests and are processed first.

bias = 3 means the drive has Read Only bias. This indicates the drive only services read requests. Write requests would have to be serviced by a different drive.

bias = 4 means the drive has Write Only bias. This indicates the drive only services write requests. Read requests would have to be serviced by a different drive. If your application only writes to one volume at a time, this guarantees that the disk is not swapped out.

### Example

```
APIRET rc;
int Bias;
```

```
Example: getting drive 0 bias:
rc = (APIRET) VIM_get_drive_bias (0,  &Bias);
Example: getting drive 1 bias:
rc = (APIRET) VIM_set_drive_bias (1,  &Bias);
```

## VIM_get_file_info

This call returns the file record for the file associated with the open file handle. This call can be used to return a variety of file information that is associated with open files such as file size, file attributes, time/date, file name, and so on.

### Syntax

```
int ENTRY VIM_get_file_info (int handle,
   ofs_records PTR file_record)
```

### Input

`handle` The handle returned by a prior successful open/create call.

`file_record` Pointer to the file_rec structure described in "File Record" on page 30.

### Example

```
APIRET rc;
ofs_records OFSrec;
rc = NO_ERROR;
rc = (APIRET) VIM_get_file_info (GlobalFileHandle,
   &OFSrec);
```

NOTE: GlobalFileHandle is assumed to be a valid file handle obtained from an prior open call.

## VIM_get_hardware_list

The GUI uses this call extensively to mount, unmount, and format volumes and to establish Library settings. This call provides status information about the drives, libraries, and volumes within the system. With this information, you can determine which hardware is managed by InveStore. You can also determine how many volumes are managed by the system and where they are located.

To use this call, perform these steps in the following order to avoid kernel terminates with an access violation.

1. Call VIMgetHardwareListSize to determine the size of this packet.
2. Allocate the DeviceStatusSize of memory indicated.
3. Call VIM_get_hardware_list, passing the buffer of size DeviceStatusSize.

### Example

```
APIRET rc;
rc = (APIRET) VIMgetHardwareListSize
   (&DeviceStatusSize);
   BaseAddressDSP = malloc(DeviceStatusSize);
   if (BaseAddressDSP)
   {
      dsp = (DEVICE_STATUS_PACKET *) BaseAddressDSP;
     dsp->DeviceStatusSize = DeviceStatusSize;
      rc = (APIRET) VIM_get_hardware_list (dsp);
   }
```

For more information, see “VIM Programming Examples” on page 110. Fully functional source code and header files are found in PDT/VIM directory after you’ve installed InveStore.

### Syntax

```
int ENTRY VIM_get_hardware_list
   (DEVICE_STATUS_PACKET PTR ListBuffer)
```

### Output

`ListBuffer` is the address of buffer that receives the returned information. If successful, the following DEVICE_STATUS_PACKET is returned.

```
/* operation request packet for data transfer element status */

typedef struct tagTRANSFER_STATUS_PACKET
{
   ULONG DeviceStatusSize;       /* total size of status structure */
   USHORT MajorStatus;               /* major status of operation */
   USHORT MinorStatus;               /* minor status of operation */
   USHORT NumLibraries;           /* number of libraries container */
   LIBRARY_STATUS LibraryStatus;        /* library container array */
```

```
   USHORT NumDataTransfers;      /* total number of data transfers */
   DATA_TRANSFER_STATUS
   DataTransferStatus;              /* data transfers array status */
} DEVICE_STATUS_PACKET;
```

`DeviceStatusSize` As stated earlier, the hardware list packet is a structure of variable size. The size of the structure changes based on the hardware attached to the system. This field defines the total size of the packet. Your buffer must be AT LEAST this large or the InveStore kernel terminates with an access violation.

`MajorStatus` A value is returned in this member whether the call is successful or not. The following values are valid:

```
OHIM_NOT_INITIALIZED
OHIM_SEM_REQUEST_ERR
OHIM_NO_DRIVE
NO_ERROR
```

`MinorStatus` Returns the number of data transfer units (drives) managed by InveStore, but is not recommended for use. Use NumDataTransfers to get this information, because this field is subject to change.

`NumLibraries` Number of library units within the system.

`LibraryStatus` An array of LIBRARY_STATUS structures.

`NumDataTransfers` Returns the number of data transfer units (drives) managed by InveStore.

`DataTransferStatus` An array of DATA_TRANSFER_STATUS structures.

`LibraryStatus` This member returns information about the libraries contained within the system. InveStore recognizes three types of libraries:

**standalone drive** A library with one drive unit and one storage slot that happens to be online. It does not have an import/export slot and does not support trapped volumes or preload initialize.

**jukebox** A library with *n* storage slots and *m* drive units, and at least one import/export slot. Supports trapped

volumes and preload initialize. Only one disk can be imported or exported at a time. You must place the disk into the jukebox after issuing the mount or format command, unless a trapped volume is being mounted or formatted.

**media changer** A library with n storage slots and 1 drive unit. It does not have an import/export slot. Supports disk packs only. Import and export operations are carried out against the entire disk pack (multiple volumes). Supports preload initialize. Only trapped volumes can be formatted in the rapid changer.

The `DEVICE_STATUS_PACKET` member Library Status is defined as follows:

```
typedef struct tagLIBRARY_STATUS
{
   SHORT LibraryType;                          /* type of container */
   USHORT LibraryStatus;        /* current library status; not used*/
   DEVICE DESC Description                   /* device description */
   VID MinimumVol;                 /* first logical volume number */
   VID MaximumVol;                  /* last logical volume number */
   USHORT NumVolumes;              /* number of volumes in library */
   UCHAR *VolStatus;        /* num of volumes in size status array */
   USHORT FirstDTnum;          /* DT number of first DT in library */
   USHORT NumDataTransfers; /* number of data transfers in library */
   struct tagDATA TRANSFER_STATUS *DataTransferStatus;   /* number of
                                                 data transfers array */
} LIBRARY_STATUS;
```

`LibraryType` The following values are valid for the LibraryType member of LIBRARY_STATUS:

```
#define CLASS_UNKNOWN 0
#define CLASS_JUKEBOX 1 //Normal jukebox
#define CLASS_STANDALONE 2
#define CLASS_MEDIACHANGER 3 //Rapid Changer
```

`MinimumVol` The starting logical volume number for this library. For Library 0, this number would be 0. The number corresponds to a storage slot. With double-sided media, there are two logical volumes associated with storage slot. Single-sided media, such as CD-ROM or media that logically appears to be single-sided have a 1-to-1 correspondence between the storage slot number and the logical volume.

`MaximumVol` Last logical volume number that is valid for this library.

`VolumeStatus` A byte array of status for each volume within the library. From this status information, you can determine whether the volume exists and where the volume currently resides. The high nibble is a set of flags which define the location/status of the volume. The low nibble is used to define which drive, within the parent jukebox contains the volume. The low nibble is only set when the high nibble has the IN_DATA_XFER flag set.

Valid Values for this field are:

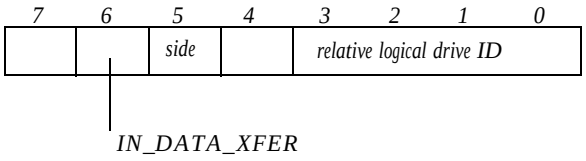
VOLUME_HOME (0x80) = Volume resides in its home storage slot within the library.

VOLUME_ABSENT (0x00) = Volume does not exist within Library. The home slot is empty.

VOLUME_UNUSABLE (0x10) = There is something wrong with the HOME Storage slot or there is a limitation in the JUKEBOX hardware/firmware that prevents this storage slot from being used. For example, a CD jukebox requires that a caddy be present in order to store media to that slot.

`IN_DATA_XFER` When this flag is set, the disk represented by this volume is within a disk drive. When this flag is set, the Low order nibble contains the relative logical drive ID of the drive that contains the media. Add FirstDTnum to this value to get the logical drive ID as it used by InveStore.

Interpret the IN_DATA_XFER bits as follows:




The IN_DATA_XFER flag has the following definition:

```
#define IN_DATA_XFER 0x40
```

Use the following mask to obtain the drive IDs.

```
#define DATA_XFER_MASK          0x0f
```

The sides of the disk are defined as follows:

```
#define SIDE_A 0x00
#define SIDE_B 0x20
```

`FirstDTnum` Starting value for Logical Drive ID that is valid for this Library. Drive ID's start at 0; therefore, the second jukebox in a system with two four drive juke-boxes would have a FirstDTnum of 4.

`NumData Transfers` The number of drives contained within this library unit.

`DataTrans-ferStatus` The DataTransferStatus member of LIBRARY_STATUS is of type DATA_TRANSFER_STATUS, which is defined as follows:

```
typedef struct tagDATA_TRANSFER_STATUS
{
   DID DataTransfer;                        /* data transfer number */
   USHORT ElementStatus;                  /* current transfer status */
   VID MinimumVol;           /* first logical volume accessible by */
                                    /* this data transfer element */
   VID MaximumVol;          /* last logical volume accessible by this
                                      data transfer element */
   USHORT ParentLibraryNum;                /*logical number of parent
                                                          library*/
   LIBRARY_STATUS ParentLibrary;      /* parent library container for
                                                           drive */
   DEVICE_DESC Description;                  /* device description */
}DATA_TRANSFER_STATUS;
```

`Data-Transfer` The 0-based drive ID associated with this drive.

`Element-Status` Defines the state of this drive. Valid values are:

```
OHIM_DRIVE_ONLINE 0036
OHIM_NO_DISK 0023
OHIM_DRIVE_OFFLINE 0037
```

`MinimumVol` Same as "MinimumVol" on page 81. From this member, you can determine who the parent library is and what slot address is valid (for a move to this drive).

`MaximumVol` Last logical volume number that is valid for this library.

`Description` The Description member of DATA_TRANSFER STATUS returns basic SCSI information about an optical hardware device. The returned information is equivalent to that returned by the SCSI INQUIRY Command (12H), which includes VendorID, ProductID, and ProductRev. Description is of the following type tagDEVICE_DESC:

`Parent-Library-Number` The logical ID of the parent library. This member can be used to access the LIBRARY_STATUS array.

`Parent-Library` Pointer into the LIBRARY_STATUS array. Points to the LIBRARY_STATUS structure of the parent library.

```
typedef struct tagDEVICE_DESC
{
   CHAR VendorID[9];                              /* device vendor */
   CHAR ProductID[17];                              /* device name */
   CHAR ProductRev[5];                    /* device revision number */
   CHAR ScsiID[3];                              /* device's SCSI ID */
   CHAR LogicalUnit[3];             /* device's logical unit number */
   CHAR ProductType[17];                          /* device's type */
   SHORT AttachedUnits;     /* number of data transfers in jukebox */
   ULONG DeviceAttribute;          /* Device attribute information */
} DEVICE_DESC;
```

The ProductType and DeviceAttribute members of tagDEVICE_DESC are further defined as follows:

`ProductType` One of the following text descriptions of the device:

"Erasable Drive"
"CDROM Drive"
"WORM Drive"
"Media Changer"
"Unknown"

`Device-Attribute` The device attribute field contains the following information about the device:

Byte 0—Used for providing media definition. For the definition of flags, see mediadef.h.

Byte 1—Provides information about the LUN (logical unit number) mode of operation of the drive. Use this field when you have a drive and jukebox on the same SCSI drive or have a single hardware device, presented as two separate devices, when it is actually just one physical device.

| *Bit* | *Description* |
|---|---|
| 16 | Indicates whether Data Transfer is LUN or not |
| 15 | Bits9 contains the LUN number for Data Transfer |
| 2-3 | Not used |

The following definitions of data attributes exist:

```
#define DATTR_MEDIATYPE_MASK (0xFF)
#define DATTR_LUN_MODE (0x00008000)
#define DATTR_LUN_MASK (0x00007F00)
```

## VIMgetHardwareListSize

The GUI uses this call to allocate memory for the hardware list. This call determines the size of the packet with hardware information. The size is needed to call VIMsetHardwareList. To use this call, perform steps in the following order to avoid an access violation.

1. Call VIMgetHardwareListSize to determine the size of this packet.
2. Allocate the DeviceStatusSize of memory indicated.
3. Call VIM_get_hardware_list, passing the buffer of size DeviceStatusSize.

For more information, see “VIM Programming Examples” on page 110.

### Syntax

```
int ENTRY VIMgetHardwareListSize
   (PULONG DeviceStatusSize)
```

### Output

If successful, the size of the VIM_get_hardware_list packet is returned in `DeviceStatusSize`.

## VIM_get_object_attributes

This call is used to retrieve the attributes for a volume, subdirectory, or file object.

### Syntax

```
int ENTRY VIM_get_object_attributes
   (pTEXT object_name,
   CurDirInfo PTR cur_directory,
   pATTRIBUTE32 attributes)
```

### Input

`object_name` The name of object (with or without path information) for which the attribute values are desired.

`cur_directory` Supplies the current directory information associated with this thread.

### Output

If successful, the function returns 0 and the attributes are stored in attributes.

## VIM_get_object_info

This call returns the file, subdirectory, or volume control record that is associated with `object_name`. It can be used to return a variety of information to the caller such as size, attributes, permissions, and so on.

### Syntax

```
int ENTRY VIM_get_object_info (pTEXT object_name,
   CurDirInfo PTR cur_directory,
   ofs_records PTR object_record)
```

### Input

`object_name` The name of the object whose record you want to have retrieved.

`cur_direct-ory` The current directory information.

### Output

If successful, the object is returned in `object_record`. For more information, see “File Record”, “Subdirectory Record”, and “Volume Record” on pages 30–31.

### Example

```
APIRET rc;
ofs_records OFSrec;
CHAR szFileName[256];
CurDirInfo LocalCurDir;

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szFileName, "\\MYVOLUME
   \\MYSUB1\\FILE.IMG");
rc = NO_ERROR;
rc = (APIRET) VIM_get_object_info (szFileName,
   &LocalCurDir, &OFSrec);
```

## VIM_get_time_date

This call returns the time and date stamp for the file based on its open handle.

### Syntax

```
int ENTRY VIM_get_time_date (int file_handle,
   pUSHORT time,
   pUSHORT date)
```

### Input

`file_handle` The handle that was returned for a prior successful open\create call.

### Output

If successful, the time and date are returned in time and date.

### Example

```
FDATE fdate;
FTIME ftime;
APIRET rc;
rc = NO_ERROR;
```

```
rc = (APIRET) VIM_get_time_date (GlobalFileHandle,
   (PUSHORT) &ftime,
   (PUSHORT) &fdate);
if (rc == NO_ERROR)
{
   printf("%02d/%02d/%02d  %02d:%02d:%02d\r\n",
      fdate.month, fdate.day, fdate.year+80,
      ftime.hours, ftime.minutes, ftime.twosecs*2);
```

## VIM_get_volume_info

The GUI uses this call for the Volume Details and Reserved Space dialog boxes. This call returns information about the desired volume.

### Syntax

```
int ENTRY VIM_get_volume_info
   (pTEXT volume_name,
   pUSHORT sector_size,
   pULONG total_sectors,
   pULONG free_sectors,
   pULONG user_reserved,
   pULONG system_reserved)
```

### Input

`volume_name` The name of volume for which information is desired. If the volume does not exist, an error is returned.

### Output

`sector_size` Contains the physical sector size of the volume.

`total_sectors` Contains the total number of sectors for this volume.

`free_sectors` Contains the number of available sectors on the volume.

`user_reserved` Describes the number of sectors that have been reserved by the user for deletes, and so on.

`system_reserved` Describes the number of sectors reserved by InveStore for internal bookkeeping.

## VIM_initialize

This call is responsible for initializing the communications channel with the optical file server. This call must be made prior to performing any I/O requests.

### Syntax

```
int ENTRY VIM_initialize (void)
```

## VIM_is_valid_path

This boolean call is used to determine if a given path is valid or invalid.

### Syntax

```
int ENTRY VIM_is_valid_path (pTEXT pathname,
   CurDirInfo PTR cur_directory)
```

`pathname` The path string you want to have validated.

`cur_directory` Contains the current directory information.

### Output

If the path exists, this function returns 1 (TRUE) or 0 (FALSE).

### Example

```
APIRET rc;
CHAR szPathName[256];
CurDirInfo LocalCurDir;

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szPathName, "\\MYVOLUME\\MYSUB1\\MYSUB2");
rc = NO_ERROR;
rc = (APIRET) VIM_is_valid_path (szPathName,
   &LocalCurDir);
```

## VIM_mkdir

This call is used to make directories. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_mkdir (pTEXT dir_name,
   CurDirInfo PTR cur_directory)
```

### Input

`dir_name` The name of the desired new directory. `dir_name` may contain path information.

`cur_directory` Supplies the kernel with information on the current directory.

### Output

Return code, 0 = success

### Post-Conditions

If successful, a new subdirectory exists on the specified volume.

### Example

```
APIRET rc;
CHAR szPathName[256];
CurDirInfo LocalCurDir;

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szPathName, "\\MYVOLUME\\MYSUB1\\MYSUB2");
rc = NO_ERROR;
rc = (APIRET) VIM_mkdir (szPathName, &LocalCurDir);
```

## VIM_mount

The GUI uses this function for the Mount dialog box and the Mount Volume item on the File menu. This function is used to import new volumes into the system. This interface is generally not an OS-transparent operation.

### Syntax

```
int ENTRY VIM_mount (pTEXT volume_name,
   DID device_id,
   int mount_type,
```

```
   int file_system_type)
```

Input

`volume_name` The name of the new volume. Usually, you set the name to asterisk (*) to mount whatever volume is loaded. Set the name to a specific volume to mount only that volume. If the imported disk does not contain the volume of the name `volume_name`, the mount terminates with an error.

`device_id` A logical number from 0 to *n*-1, where *n* is the number of drives being managed by the OHIM. This field indicates which drive the volume should be mounted in. In the case of a jukebox there is no guarantee that the drive specified is the one used. The Volume Manager picks the best drive to use, based on a number of criteria.

The device_id for the mount, unmount, and format commands is mainly used to determine for which library the call is intended. Each Library has a range of valid drive IDs. For example, the first Library on the BUS with 2 drives would have drive IDs 0 and 1; a second, two drive library on the BUS would have IDs 2 and 3. Typically, you set the drive ID to the first valid drive for the intended library. The library uses the selected drive if possible, so don't bother trying to control the disk placement.

`mount_type` Specifies whether this is a full physical mount or a cache-file-only type mount operation. `mount_type` has the following values:

1 = Standard mount

2 = Mount volume with no output; don't send any mount information to the calling application or to screen.

3 = Read only mount. Volume can't be modified.

4 = Directory only mount. This allows off-line volumes to be searched.

`file_system_type` Not currently used.

### Output

Return code, 0 = successful

**Note** If the cartridge is double-sided, the volume on the other side is also mounted if the container is a jukebox.

### Example

```
APIRET rc;
DID usDID;
CHAR szVolser[256];
strcpy (szVolser, "\\*");
usDID=0; // mount volume into first Library
rc = (APIRET) VIM_mount(szVolser,
   (DID) usDID,
   (int) MOUNT_VOL,
   0);
```

## VIM_open

This call is used to create/open the requested file. This is a general purpose call to open a file or to create a file based on the flags parameters.

### Syntax

```
int ENTRY VIM_open (pTEXT file_name,
   CurDirInfo PTR cur_directory,
   pNINT file_handle,
   pULONG pulAction,
   ATTRIBUTE32 attributes,
   ULONG ulFlag,
   ULONG ulMode)
```

### Input

`file_name` The name of the file (with or without volume/path information).

`cur_directory` Contains the kernel specific current directory information.

`mode` Defines the access rights of the file such as read only, read-write, write-only.

### Output

`file_handle`. If successful, the open file handle token is returned in this parameter.

`pulAction` The action taken by the kernel is returned in this parameter. It can take the following values:

1 = the file existed and was open.

2 = the file didn't exist and was opened.

3 = the file existed and was truncated.

`attributes` This parameter contains the attributes of the file if it is created. For more information, see "Common Fields" on page 28.

`ulFLAG` This flag describes the actions to be taken based on whether the file exists or not. Only the low byte of the flag is used. The low nibble of the byte describes the action to be taken if the file exists. The high nibble describes the actions to take if the file doesn't exist. The values are as follows:

- low nibble (4 bits) if file exists
- 0x0000 = fail operation if file exists
- 0x0001 = open file if it exists
- 0x0002 = replace (overwrite) file if it exists
- high nibble (4 bits) if file doesn't exist
- 0x0000 = fail operation if file doesn't exist
- 0x0010 = create file if it doesn't exist

`ulMode` This flag describes the type of access being requested. The values are as follows:

- 0x0000 = read only access
- 0x0001 = write only access
- 0x0002 = read write access

### Example

```
APIRET rc;
CHAR szFileName[300];
CurDirInfo LocalCurDir;
int LocalHandle;
ULONG ulAction;
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szFileName, "\\MYVOLUME\\FILE.IMG");
rc = (APIRET) VIM_open(szFileName,
   &LocalCurDir,
```

```
&LocalHandle,
&ulAction,
(ATTRIBUTE32) ATTRIB_FILE,
 VOPEN_ACTION_OPEN_IF_EXISTS,
 VOPEN_ACCESS_READONLY);
```

## VIMopenCreate

This new call resembles the WIN32 version of the call. With this call you can specify the location, read/write access, sharing, size, and attributes of a file.

### Syntax

```
VIMOpenCreate ( PCHAR  Filename,
   CurDirInfo * pCurrentDir,
   ULONG uAccessFlags,
   ULONG uShareMode,
   ULONG uCreateDisp,
   UINT32 uAttributes,
   UINT64 uInitialSize,
   PULONG puAction,
   PNINT pFileHandle )
   nTaken, pHANDLE FileHandle);
```

### Input:

`Filename` Name of file to be opened or created. May include fully qualified path.

`CurDirInfo` Current directory pointers.

`uAccess-Flags` Defines the type of access you want to grant to this file with one or a combination of the following values:

| *Value* | *Mnemonic* | *Comment* |
|---|---|---|
| 0x00001 | READ_ACCESS | Set both bits for read-write access. |
| 0x00002 | WRITE_ACCESS | |
| 0x00010 | WRITE_THROUGH_ CACHE | Commits data stored in cache to optical when the file is closed. |

`uShareMode` The following flags define how the file may be accessed by others:

| *Value* | *Mnemonic* | *Comment* |
|---|---|---|
| 0x0000 | EXCLUSIVE_ACCESS | Nobody else may access the file until the file is closed; only a single instance of the file can be opened. |
| 0x0001 | FILE_SHARE_READ | Anyone else may open the file to read it. |
| 0x0002 | FILE_SHARE_WRITE | Not supported. |

You can specify a single flag or the logical sum of the FILE_SHARE flags.

`uCreateDisp` The following flags define the process for opening files:

| *Value* | *Mnemonic* | *Comment* |
|---|---|---|
| 1 | CREATE_NEW | Create file if it doesn't exist. Fail if it exists. |
| 2 | CREATE_ALWAYS | Create File if it doesn't exist. Overwrite (destroy) existing file if it does. |
| 3 | OPEN_EXISTING | Open file if its exists. Fail if it doesn't exist. |
| 4 | OPEN_ALWAYS | Open file if it exists. Create it if it doesn't. |
| 5 | TRUNCATE_EXISTING | Set file size to 0 if file exists. Fail if it doesn't exist. |

Specify exactly one of the these flags.

`uAttributes` The following attributes are defined:

The following table lists the possible values for the attribute field of a record returned by an OFS call:

| *Attribute* | *Description* |
|---|---|
| ATTRIB_READ_ONLY | The file cannot be deleted or updated. This field retains this meaning under any operating system. |
| ATTRIB_HIDDEN | The file is hidden from normal directory searches. |
| ATTRIB_SYSTEM | The file is hidden from normal searches. |
| ATTRIB_VOLUME | This allows a user name to be placed on a file system volume. Entry lives in the root directory only. If this bit is used for a search parameter, the search is exclusive. This field retains this meaning under any operating system. |
| ATTRIB_SUBDIR | Indicates that this file is a special subdirectory entry, excluded from normal searches. This field retains its meaning under any operating system. |
| ATTRIB_ARCHIVE | Indicates an file archive. |
| ATTRIB_FILE | Indicates an ordinary file with no special properties. Generally used for text, data, and applications programs. This field retains this meaning across all operating systems. |

`InitialSize` Specifies an initial size for the file being created. This parameter is very useful for providing preallocation for data files, especially on WORM media for keeping data files contiguous. This parameter is ignored for existing files. If this parameter is 0 then no initialize size is specified.

## Output

`uAction` The action taken by the kernel is returned in this parameter. It can take the following values:

VOPEN_FILE_EXISTED 1—the file existed and was open.

VOPEN_FILE_CREATED 2—the file didn't exist and was opened.

VOPEN_FILE_TRUNCATED 3—the file existed and was truncated.

`pFileHandle` The arbitrary token for making subsequent references to the file.

### Example

```
APIRET rc;
CHAR   in_char;
CHAR   szFileName[256];
UINT64 initialSize;
ULONG  ulActionTaken;
ULONG  ulAccessFlag;
ULONG  ulCreateDisposition;
ULONG  ulOpenFlag;
ULONG  ulShareMode = 0;
CurDirInfo LocalCurDir;
int LocalFileHandle;
rc = NO_ERROR;
initialSize.Lower = 0;
initialSize.Upper = 0;
ulAccessFlag = 0;
ulCreateDisposition = 0;
ulOpenFlag = 0;
strcpy (szFileName, "\\MYVOLUME\\FILE.IMG");
ulAccessFlag = 0;                       // LAZY WRITE
ulAccessFlag |= READ_ACCESS | WRITE_ACCESS;
ulCreateDisposition = CREATE_NEW;
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
rc = (APIRET) VIMopenCreate( szFileName, // fileNa-
me
   &LocalCurDir,                        // currentDir
   ulAccessFlag,
   ulShareMode,
   ulCreateDisposition,
   (ATTRIBUTE32) ATTRIB_FILE,
   initialSize,
   &ulActionTaken,
   &LocalFileHandle );
```

## VIM_read

This call reads data from the open file.

### Syntax

```
int ENTRY VIM_read(int file_handle,
   pUCHAR buffer,
```

```
   IOSIZE count,
   pIOSIZE return_count)
```

### Input

`file_handle` The handle returned by a prior successful open/create call.

`buffer` The caller's data buffer.

`count` The number of bytes to read.

### Output

`return_count` Contains the number of bytes successfully read.

### Example

```
APIRET rc     = NO_ERROR;
IOSIZE ToRead = 0;
IOSIZE WeRead = 0;
PVOID  pBuff  = NULL;
ToRead = 32768;                          // read 32K
pBuff = malloc (ToRead);
memset(pBuff, 0x00, (size_t) ToRead);
rc = (APIRET) VIM_read (GlobalFileHandle,
   pBuff,
   (IOSIZE) ToRead,
   (pIOSIZE) &WeRead);
```

## VIM_rename

The GUI uses this call for the Rename Volume item on the File menu. This call is used the rename files, directories, subdirectories, and volumes. This function is not applicable to the CD/DVD-ROM software.

**Note** You can use this call to rename a CD volume. The call logically renames the volume; the new name exists in cache only. When the cache is deleted the volume name reverts back to the original name.

### Syntax

```
int ENTRY VIM_rename (pTEXT old_name,
   pTEXT new_name, CurDirInfo PTR cur_directory)
```

### Input

`old_name` The current name of the object.

`new_name` The desired name of the object.

Both `old_name` and `new_name` may or may not contain volume/path information.

`cur_directory` Contains the kernel-specific, current directory data.

### Example

This example shows how to rename a volume.

```
APIRET rc;
CHAR szOldName[256];
CHAR szNewName[256];
CurDirInfo LocalCurDir;
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szOldName, "\\MYVOLUME");
strcpy (szNewName,"\\NEWVOLNAME");
rc = (APIRET) VIM_rename (szOldName, szNewName,
   &LocalCurDir);
```

## VIM_rmdir

This call is used to remove a directory or volume. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_rmdir (pTEXT dir_name,
   CurDirInfo PTR cur_directory)
```

### Input

`dir_name` The name of the directory to remove. `dir_name` may contain path information.

`cur_directory` Supplies the kernel with information on the current directory.

### Pre-Conditions

The directory to be removed must be empty (have no children). Directories that contain files or directories cannot be removed.

### Post-Conditions

If successful, the specified directory no longer exists on the specified volume.

### Example

```
APIRET rc;
CHAR szPathName[256];
CurDirInfo LocalCurDir;

memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szPathName, "\\MYVOLUME\\MYSUB1\\MYSUB2");
rc = NO_ERROR;
rc = (APIRET) VIM_rmdir (szPathName, &LocalCurDir);
```

## VIM_search

The GUI uses this call for the Refresh and Volume Details functions. This call is used to search for files, subdirectories, and volumes. To find all the volumes known to the system, simply set the search name to "*.*". The call searches the root of InveStore and returns the volume records for the system. When you search a subdirectory in the root of InveStore, a volume is returned if a match is made.

### Syntax

```
int ENTRY VIM_search (pTEXT search_name,
   search_rec PTR search_info,
   ofs_records PTR control_rec,
   CurDirInfo PTR cur_directory)
```

### Input

`search_name` The search specifications (with or without volume/path information). The search name may contain wildcards.

`search_info` The kernel-specific search information. This struct should be set to zero for a search first operation.

`cur_directory` The kernel-specific current directory information attributes for the search.

### Output

`search_info`, upon a successful search, is filled with information for a search next operation and should be used for the search next call. `search_rec` has the following structure:

```
typedef struct
   {
    USHORT reserved[7];
    ATTRIBUTE32 search_attributes;
   } search_rec;
```

`control_rec` is filled with the file, volume, or subdirectory record from a successful search. “File Record”, “Subdirectory Record”, and “Volume Record” on pages 30–31.

### Pre-Conditions

If `search_info` is not zero, `cur_directory` is not needed. The path portion of `search_name` is likewise not needed for a search next.

### Example

```
search_rec  InSearchInfo;
ofs_records OFSrec;
APIRET rc;
UCHAR szInSearch[512];
CurDirInfo LocalCurDir;
strcpy( szInSearch, "\\*.*" );
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
   LocalCurDir.volume_id = NO_VOLUME;
memset(&OFSrec, 0x00, sizeof(OFSrec));
memset(&InSearchInfo, 0x00, sizeof(search_rec));
InSearchInfo.search_attributes = (ATTRIBUTE32)
   ATTRIB_SUBDIR | ATTRIB_ARCHIVE;|
rc = (APIRET)VIM_search (szInSearch,
   &InSearchInfo, &OFSrec, &LocalCurDir);
```

## VIM_set_drive_bias

The GUI uses this call to set the drive bias in Library Settings function on the Edit menu. This call allows you to set the drive bias. The drive bias decides what type of I/O requests receive priority on a given device. There are five different types of I/O bias which may be set for a drive. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_set_drive_bias (DID DeviceId,
   int bias)
```

### Input

DeviceId

A logical number from 0 to *n*-1 where *n* is the number of drives being managed by the OHIM.

`bias` Contains the desired bias. The values of bias are defined as follows:

bias = 0 means the drive has no bias. This indicates that all I/O requests are treated equally and are processed in an FIFO manner.

bias = 1 means the drive has Read Over Write bias. This indicates that read requests have priority over write requests and are processed first.

bias = 2 means the drive has Write Over Read bias. This indicates that write requests have priority over read requests and are processed first.

bias = 3 means the drive has Read Only bias. This indicates the drive only services read requests. Write requests would have to be serviced by a different drive.

bias = 4 means the drive has Write Only bias. This indicates the drive only services write requests. Read requests would have to be serviced by a different drive. If your application only writes to one volume at a time, this guarantees that the disk is not swapped out.

### Output

Return code, 0 = success

### Example

```
APIRET rc;
int Bias;
Example: setting drive 1 bias to write only:
Bias =  4;
rc = (APIRET) VIM_set_drive_bias  (1,  Bias);
```

## VIM_set_drive_status

The GUI uses this call to set the drive status in the Library Settings function on the Edit menu. This call is used to set the status of a drive. The status referred to is whether the drive is to be considered on-line and accessible or off-line and not usable. This call is to allows an operator to flag a drive that appears to be having hardware problems as being off-line, so it is not used.

### Syntax

```
int ENTRY VIM_set_drive_status (DID drive,
   int status)
```

### Input

`drive` is a logical number from 0 to *n*-1, where *n* is the number of drives being managed by the OHIM. This device ID specifies which device the call is intended for.

*status* specifies the status of the drive:

1 = drive is to be off-line and not usable.

2 = drive is to be on-line and usable.

### Output

Return code, 0 = success

### Post-Conditions

If successful, the status of the intended drive has been changed.

## VIM_set_object_attributes

This call changes the attributes of a file, subdirectory, or volume object. This function is not applicable to the CD/DVD-ROM software.

```
Syntax
int ENTRY VIM_set_object_attributes
   (pTEXT object_name,
   CurDirInfo PTR cur_directory,
   ATTRIBUTE32 new_attributes)
```

### Input

`object_name` Name of file or subdirectory with or without volume/path information.

`cur_directory` Contains the kernel specific current directory information.

`new_attributes` Contains the file attributes desired for object. For more information, see “Common Fields” on page 28.

### Output

Return code, 0 = success

### Example

This example sets a files attributes to read only and hidden.

```
APIRET rc;
CHAR szObjectName[256];
ATTRIBUTE32 attrib;
CurDirInfo LocalCurDir;
memset(&LocalCurDir, 0x00, sizeof(CurDirInfo));
LocalCurDir.volume_id = NO_VOLUME;
strcpy (szObjectName, "\\MYVOLUME
  \\MYSUB1\\FILE.IMG");

attrib = ATTRIB_READ_ONLY | ATTRIB_HIDDEN;
rc = (APIRET) VIM_set_object_attributes
   (szObjectName, &LocalCurDir, attrib);
```

## VIM_set_time_date

This call is used to change the time/date stamp of the file associated with handle. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_set_time_date (int handle,
   USHORT time,
   USHORT date)
```

### Input

`handle` was returned by a prior successful open/create call.

`time` and `date` are the new time/date stamps for the file. For more information, see "Common Fields" on page 28.

### Output

Return code, 0 = success

### Post-Conditions

If successful, the time/date stamp of the file is changed.

### Example

```
FDATE fdate;
FTIME ftime;
APIRET rc;
struct tm *DateTime;
time_t timeVal;
USHORT usDate, usTime;

rc = NO_ERROR;
time(&timeVal);
DateTime = localtime(&timeVal);
// localtime gets 0-12, 0 = January
fdate.month = (USHORT) DateTime->tm_mon + 1;
fdate.day   = (USHORT) DateTime->tm_mday;
// localtime gets year-1900 but
//fdate wants year-1980
fdate.year  = (USHORT) DateTime->tm_year - 80;
ftime.twosecs = (USHORT) DateTime->tm_sec / 2;
ftime.minutes = (USHORT) DateTime->tm_min;
ftime.hours   = (USHORT) DateTime->tm_hour;
memcpy(&usDate, &fdate, sizeof(USHORT));
memcpy(&usTime, &ftime, sizeof(USHORT));
rc = (APIRET) VIM_set_time_date (GlobalFileHandle,
   usTime, usDate);
```

## VIM_set_volume_attr

This call is used to set the 44-byte extended attribute field of the volume record. This extended attribute permits applications to record information about the contents of the volume and other pertinent data. It provides a fast method for querying the contents of a library, since the volume information is cached to the hard disk. *No* disks need to be loaded to obtain this attribute data. The privilege attribute may also be changed.

### Syntax

```
int VIM_set_volume_attr (PCHAR VolumeName
   ATTRIBUTE32 PrivilegeAttributes,
   PCHAR ExtendedAttribute);
```

### Input

`VolumeName` The name of the volume you use to add or change the extended attribute. This string should only contain the volume name itself.

`Privilege Attributes` Described in the volume record. It is not currently used, but it is recorded. For more information, see "Volume Record" on page 31.

`Extended Attribute` 44 bytes of user information. The buffer must be 44 bytes long.

### Output

Return code, 0 = success. The privileges attribute and the extended attribute are updated with this call.

### Example

```
APIRET rc;
CHAR szVolser[256];
CHAR szExtendedAttribs[44];
ATTRIBUTE32 attrib;
strcpy (szVolser, "\\MYVOLUME");
strcpy (ExtendedAttribs, "Accounting Dept
   Records for Dec 1997");
// note volume priveledge attributes
// are not enforeced
attrib = READ_VOLUME_DATA  | WRITE_VOLUME_DATA
   | SEARCH_VOL_FILES;
rc = (APIRET) VIM_set_volume_attr (szVolser,
    attrib, szExtendedAttribs);
```

## VIM_set_volume_info

This call sets the number of reserved user sectors. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_set_volume_info
   (pTEXT volume_name, ULONG user_reserved)
```

### Input

`user_reserved` Contains the number of sectors to reserve.

### Output

Return code, 0 = success

### Example

```
APIRET rc;
CHAR szVolName[256];
ULONG ulUserReserved;
strcpy (szVolName, "\\MYVOLUME");
ulUserReserved = 2000;      // Reserve 2000 sectors
rc = (APIRET) VIM_set_volume_info (szVolName,
   ulUserReserved);
```

## VIM_shutdown

The GUI uses this call for the Reserved Space Settings function on the Edit menu. This call is responsible for bringing the kernel to an orderly shutdown. Disks within a jukebox are returned to their home shelf. All cache buffers are flushed. Any open file handles are closed. The optical hardware is brought to a power-down ready state. This module calls all other shutdown modules within the kernel.

### Syntax

```
int ENTRY VIM_shutdown (void)
```

## VIM_truncate_file

This call truncates the size of the file at the current file pointer position. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_truncate_file (int handle)
```

### Input

`handle` Was returned by a prior successful open/create call.

### Output

Return code, 0 = successful.

## VIM_unmount

The GUI uses this call to implement the Unmount button on the main console and the Unmount Volume function on the file menu. This function is used to export volumes from the system. This interface is generally not OS transparent.

### Syntax

```
int ENTRY VIM_unmount (pTEXT volume_name,
   int unmount_type, DID device_id,
       pTEXT offline_location)
```

### Input

`volume_name` The name of the volume to be exported.

`device_id` A logical number from 0 to *n*-1, where *n* is the number of drives being managed by the OHIM; specifies which device the call is intended for. `device_id` is optional. If present, `device_id` instructs the Volume Manager to unmount whatever volume is in the specified drive. If `device_id` is specified, the volume name is not necessary. If the `volume name` and the `device id` are both present, the volume name must either be a wildcard (*) or it must match the volume name within the device; otherwise, an error is returned. If the cartridge is double-sided, the volume on the other side is also unmounted if the container is a jukebox.

`unmount_type` Specifies whether this is a destructive unmount (the cache files are destroyed) or a simple unmount operation (cache files are preserved). `unmount_type` has the following values:

1 = Normal unmount. The disk is removed from the drive. If the container is a jukebox, the disk is put back in the appropriate shelf. The cache files are saved.

2 = Unmount and eject. The disk is ejected from the drive. If the container is a jukebox, the disk is ejected to the export slot. The cache files are saved.

3 = Destructive unmount. The disk is ejected from the drive. If the container is a jukebox, the disk is ejected to the export slot. The cache files are destroyed. The Volume Manager has no knowledge of this cartridge.

`off-line location` A caller-supplied text string that describes the off-line location, for example “Cabinet XYZ in storage building 1023”.

### Example

```
APIRET rc;
CHAR in_char;
CHAR szVolser[256];
CHAR szOffLoc[44];
USHORT usDID;
USHORT usType;
strcpy (szVolser, "\\MYVOLUME");
strcpy(OffLoc, "Cabinent XYZ, Drawer Y");
usDID=0;                    // this parameter ignored
usType = UNMOUNT_AND_EJECT;
rc = (APIRET) VIM_unmount(szVolser,
   usType, (DID) usDID, szOffLoc);
```

## VIM_write

This call is used to write data to the optical disk. This function is not applicable to the CD/DVD-ROM software.

### Syntax

```
int ENTRY VIM_write (int file_handle,
   pUCHAR buffer, IOSIZE count,
   pIOSIZE return_count)
```

### Input

`file_handle` Token returned by a prior successful open/create call.

`buffer` A pointer to the memory location which contains the data to be written.

`count` The number of data bytes to be written.

### Output

`return_count` contains the number of bytes actually written.

### Example

```
APIRET rc     = NO_ERROR;
IOSIZE ToWrite = 0;
IOSIZE WeWrote = 0;
PVOID  pBuff  = NULL;
ToWrite = 32768;                          // write 32K
pBuff = malloc (ToWrite);
memset(pBuff, 0xac, (size_t) ToWrite;
rc = (APIRET) VIM_write (GlobalFileHandle,
   pBuff, (IOSIZE) ToWrite, (pIOSIZE) &WeWrote);
```

## VIM Programming Examples

The following example in C involves formatting a disk using a VIM call.

### Example 1

```
APIRET PerformVolumeFormat(VOID)
{
   APIRET rc;                                        /* 16-bit UINT */
   UCHAR szVolName1[50];
   UCHAR szDiskName1[50];
   UCHAR szVolName2[50];
   UCHAR szDiskName2[50];
   ULONG input;
   DID Drive;
   int Reformat;
   int MountAfterFormat;

   printf ("\r\n");
   printf ("Enter Volume Name (Side 1):\r\n");
   fflush(stdout);
   fflush(stdin);
   memset(szVolName1, 0x00, sizeof(szVolName1));
   gets(szVolName1);
   printf ("\r\n");
   strcpy(szDiskName1, szVolName1);

   printf ("\r\n");
   printf ("Enter Volume Name (Side 2):\r\n");
   fflush(stdout);
   fflush(stdin);
   memset(szVolName2, 0x00, sizeof(szVolName2));
   gets(szVolName2);
```

```
   printf ("\r\n");
   strcpy(szDiskName2, szVolName2);

   printf("Enter drive id (for library selection)... ");
   fflush(stdout);
   scanf("%lu", &input);
   printf("\r\n");
   Drive = (DID) input;

   printf("Enter 0 for do not format existing disk or 1 to allow... ");
   fflush(stdout);
   scanf("%lu", &input);
   printf("\r\n");
   Reformat = (int) input;
   if ((Reformat != 0) && (Reformat != 1))
   { printf("Value incorrect will use 0 for do not reformat.\r\n");
    Reformat = REFORMAT_FALSE;
   }

   printf("Enter 0 for Eject after format or 1 to Mount... ");
   fflush(stdout);
   scanf("%lu", &input);
   printf("\r\n");
   MountAfterFormat = (int) input;
   if ((MountAfterFormat != 0) && (MountAfterFormat != 1))
   { printf("Value incorrect will use 0 for do not mount.\r\n");
    MountAfterFormat = (int) FORMAT_MOUNT;
   }

   rc = (APIRET) VIM_format_media(szVolName1,
       szDiskName1,
       szVolName2,
       szDiskName2,
       Drive, Reformat,MountAfterFormat, 0);
   return (rc);
}
```

### Example 2

The following is an example in C of the VIMgetHardwareList call.

```
APIRET RC;
PVOID BaseAddressDSP;
DEVICE_STATUS_PACKET *DSP
DATA_TRANSFER_STATUS *DriveStat;
LIBRARY_STATUS *LibStat;
ULONG DeviceStatusSize;
USHORT usData;
USHORT usIndex
USHORT usVol;

rc = (APIRET) VIMgetHardwareListSize (&DeviceStatusSize);

if (rc == NO_ERROR)
   {
   rc = DosAllocMem(&BaseAddressDSP, DeviceStatusSize;
   PAG_WRITE | PAG_READ | PAG_COMMIT);
   }

if (rc == NO_ERROR)
   {
   dsp = (DEVICE_STATUS_PACKET * ) BaseAddressDSP;
   dsp->DeviceStatusSize = DeviceStatusSize;
   rc = (APIRET) VIM_get_hardware_list (dsp);
   }
FormatVIMReturnCode
   (“VIM_get_hrdware_list”,rc, szDisplayMsg,sizeof(szDisplayMsg));

if (rc==NO_ERROR)
{
   printf(“MajorStatus =%hu, MinorStatus =%hu\r\n”,
       dsp->MajorStatus,
       dsp->MinorStatus);
   printf(“\r\n”);
   DriveStat=dsp->DataTransferStatus

for(usData=0;usData<dsp->NumDataTransfers;usData++)
{
   printf(“DataTransferStatus:\r\n”);
   printf(“DataTransfer =%u\r\n”,
       dsp->DataTransferStatus[usData].Data Transfer);
   printf(“ElementStatus =%u\r\n”,
       dsp->DataTransferStatus[usData].ElementStatus);
   printf(“MinimumVol =%u\r\n”,
       dsp->DataTransferStatus[usData].MinimumVol);
   printf(“MaximumVol =%us\r\n”
```

```
       dsp->DataTransferStatus[usData].MaximumVol);

   printf("Parent Library Number =%d\r\n"
       dsp->DataTransferStatus[usData]. ParentLibraryNum);
for (usVol=0;
   usVol<=(dsp->DataTransferStatus[usData].MaximumVol -
       dsp->DataTransferStatus [usData].MinimumVol);
   usVol++)
{
   printf("VolStatus[%d] =%d\r\n" usVol, dsp-
       >DataTransferStatus[usData].ParentLibrary->VolStatus[usVol]);
}
   printf("r\n");
   printf("DeviceDescription:\r\n");
   printf("VendorID=%s\r\n",
       dsp->DataTransferStatus[usData].Description. VendorID);
   printf("ProductID=%s\r\n",
       dsp->DataTransferStatus[usData]. Description.ProductID);
   printf("ProductRev=%s\r\n",
       dsp->DataTransferStatus[usData]. Description.ProductRev);
   printf("ScsciID=%s\r\n",
       dsp->DataTransferStatus[usData]. Description.ScsiID);
   printf("LogicalUnit=%s\r\n"
       dsp->DataTransferStatus[usData]. Description.LogicalUnit);
   printf("ProductType=%s\r\n",
       dsp->DataTransferStatus[usData]. Description.ProductType);
   printf("AttachedUnits =%d\r\n",
       dsp>DataTransferStatus[usData]. Description.AttachedUnits);
}
```

# Appendix A

## ERROR CODES

Pegasus has attempted to document as completely as possible the error codes relating to the InveStore software. The following categories of error codes are listed in this document:


| *Error Category* | *Page* |
|---|---|
| Asynchronous Timer Errors | 157 |
| Basic Disk Errors | 157 |
| Btree Errors | 130 |
| DLL Errors | 156 |
| File Allocation Descriptor (FAD) BTREE Errors | 140 |
| File Errors | 166 |
| General Errors | 117 |
| Interprocess Communications Errors | 165 |
| Memory Manager Errors | 128 |
| Mount Error | 143 |
| OHIM Errors | 144 |
| PFM error codes | 142 |

| *Error Category* | *Page* |
|---|---|
| Semaphore Manager Errors | 159 |
| Shared Memory Errors | 166 |
| Thread/Process Errors | 158 |

Unfortunately, the entire range of codes caused by actions of third party programmers or manufacturers cannot be documented. Drive manufacturers often add or change internal error codes to reflect changes in the optical hardware and its circuitry.

## General Format of the Error Log File

The error log file contains a complete description of all system errors that occur. It also contains informational logging of events that provide the users with an audit trail of various actions that have occurred. Informational logs are not errors and should not be reported to Pegasus.

The log file is created and stored in the PDT\BIN directory or other directory where you installed InveStore. The file name consists of the letters "OJ" followed by the date on which the log file was created. The file extension is ".log". For example:

OJ020795.log is the log file that was created on February 7th, 1995. For each new day that the operator GUI console is initialized, a new log file is created.

Normal errors such as "file not found" are not recorded to the error log. Only those errors which indicate a more severe problem are recorded.

The error log file is not normally deleted and may therefore grow quite large. The only time the InveStore kernel will delete the log is if there is not enough free magnetic disk space when InveStore is initialized. Be sure that at least 10M bytes is available on the local magnetic drive at all times.

## Fatal Errors

Fatal errors as used here means errors which are of sufficient severity to render the current operation impossible to continue or complete.

# Error Code Directory

## General Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0001 | Invalid function error | An attempt was made to call a function which does not exist or is not supported. This is generally caused by a bug in the user's application. Debug the application. |
| 0x0002 | File not found error. File does not exist. | Check the spelling. |
| 0x0003 | Path not found error. Bad path specifier. | Check the spelling. |
| 0x0004 | No file handles left. | Close some files or reconfigure the HANDLES= parameter. |
| 0x0005 | Access denied error. Operation cannot be supported at this time. | This generally occurs when an illegal operation such as deleting a read-only file is attempted. |
| 0x0006 | Invalid Handle error. | An attempt was made to use an unopened or non-existent file handle. This is generally a bug in the user's application. Debug the application. |
| 0x000d | Invalid parameter. | Input parameter passed to a function call was not a valid value. This is generally a bug in the user's application. Debug the application. |
| 0x0012 | No more files. The search next has exhausted all possible matches. | Normal error received at end of search. |
| 0x0013 | File already exists. | An attempt was made to create a file or directory that already exists. This is an illegal operation. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0014 | Unable to make dir entry. Error in directory creation. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drives have been serviced and cleaned. |
| 0x0090 | Recovered a bad CDR. | The system found a bad directory record and recovered it. You may have lost a part of your file. Do not report this error to Pegasus. Contact your hardware manufacturer. |
| 0x0091 | The system found a bad data link and corrected it. | You may have lost a part of the file. Do not report this error to Pegasus. |
| 0x0092 | The system recovered a bad file size. | It corrected the file that did not match the actual recorded size. Do not report this error to Pegasus. |
| 0x009f | Disk marked full because of write error - spares table full. | The disk is not physically full, but the controller will no longer allow writes. |
| 0x00a0 | Disk full error. No more free sectors. | The disk can no longer be written to. |
| 0x00a1 | Unknown control record. Directory record cannot be recognized. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00a2 | Directory record found is not a valid File Descriptor block. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00a3 | Control record missing. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00a4 | The recorded file size does not match the number of data descriptors. | |
| 0x00a5 | Bad Data Segment Descriptor. Control record contained an invalid DSD. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00a7 | Unreadable disk parameter block. System area contains failing sectors. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00a8 | Bad system parameter block. System area contains failing sectors. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00a9 | Bad Partition parameter block. System area contains failing sectors. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00aa | Low Level Disk label unreadable. System area contains failing sectors. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00ab | File Descriptor block is missing. Directory is damaged. | This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00ac | Control Record name doesn't match File Descriptor Block. | The directory record is corrupt. This is generally caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x00ae | Disk has not been software formatted. | Issue the format command before mounting the cartridge. |
| 0x00af | Disk not PDT format. | The disk has been initialized by some other file system. Disk must be reformatted (if it is an erasable disk) before it can be used. WORM disks that contain another file system cannot be used by the Pegasus software. |
| 0x00b0 | Mag Cache/Optical Cache mismatch. | The configuration file contains invalid cache parameters. The OPTICAL.CNF file has been incorrectly set. The Pegasus kernel will use the defaults for the cache options instead. |
| 0x00b2 | The number of users in the Registry is in error and is out of range. | Reconfigure the OPTICAL.CNF. |
| 0x00b3 | Invalid number of volume specified in configuration file. | VOLUMES= specified is out of range. Configuration file contains invalid volume parameters. The OPTICAL.CNF file has been incorrectly set. The Pegasus kernel will use the defaults for the VOLUMES= options instead. |
| 0x00b4 | Bad/missing numeric value in configuration file. | The configuration file contains an invalid line that does not specify a value. The OPTICAL.CNF file has been incorrectly set. The Pegasus kernel will use the defaults for the cache options instead. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00b6 | Invalid magnetic partition size. The magnetic partition specification was invalid. | The configuration file contains an invalid magnetic partition value. The OPTICAL.CNF file has been incorrectly set. The Pegasus kernel will use the defaults for the cache options instead. |
| 0x00b8 | Bad parameter in configuration file. | The configuration file contains an invalid line that specifies an unknown parameter. The OPTICAL.CNF file has been incorrectly set. The Pegasus kernel will discard the invalid line. |
| 0x00c0 | Magnetic file open failure. Unable to open cache or spool file. | This generally occurs because of invalid settings in the OPTICAL.CNF file.<br>Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |
| 0x00c1 | Magnetic file close failure. Unable to close cache or spool file. | This generally occurs because of insufficient disk space.<br>Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00c2 | Magnetic file read failure. Unable to read desired data from cache or spool file. | |
| 0x00c3 | Magnetic file write failure. Unable to write specified bytes to cache or spool file. | This generally occurs because of insufficient disk space.<br>Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |
| 0x00c4 | Magnetic file create failure. Unable to create cache or spool file. | This is generally caused by an invalid SPOOL FILES= parameter in the configuration file or by insufficient magnetic disk space.<br>Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00c5 | Magnetic file position failure. Unable to move cache or spool file pointer to desired location. | Make sure that someone has not deleted the Pegasus cache files while the system was in operation. If this is not the case, report the error to Pegasus Disk Technologies. |
| 0x00c6 | Insufficient Magnetic disk space. Cache is exhausted or below minimum level. | You will have to free some disk space. The Pegasus kernel will not be able to write any data or make any new directories until more disk space is available. |
| 0x00c7 | Magnetic file delete failure. Unable to delete cache or spool file. | Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |
| 0x00c8 | Invalid file number. Attempt to access cache file with an invalid number. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00c9 | Magnetic cache is completely consumed. No more magnetic disk space. | Make sure your SPOOL FILES= parameter has been correctly set. If SPOOL FILES was set to use a RAM disk, make sure the RAM disk still exists.<br>Make sure that the Pegasus kernel has at least 5M bytes of free disk space.<br>Make sure the Pegasus software is not using the same drive as the operating system virtual memory disk swapper.<br>Make sure that someone has not deleted the Pegasus cache files while the system was in operation. |
| 0x00ca | Attempt to access cache file type that does not exist. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories.<br>Shut down the system and restart. |
| 0x00d2 | Volume already mounted. Attempt to mount the same volume twice. | This is a user error. Do not report this error to Pegasus. |
| 0x00d3 | Duplicate volume error. Attempt to mount a volume with the same name as an existing volume. | You have attempted to mount a cartridge that contains the same low level disk name and disk ID. This operation is illegal. This is a user error. Do not report this error to Pegasus. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00d6 | Error getting volume ID. Unable to create a virtual volume ID. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x00d7 | Disk not volume labeled. No valid name for disk. | The disk must contain a valid label. Relabel the cartridge before mounting. |
| 0x00d8 | Unknown or invalid volume. | An attempt was made to access a non-existent volume. This is the same as a BAD PATH error. Check your spelling. |
| 0x00d9 | Volume not mounted. | An attempt was made to access files on an off-line volume. Mount the cartridge if access to the data is required. |
| 0x00da | Kernel not initialized. | An attempt was made to access software before it has been initialized. You must allow the system to initialize before it can be used. |
| 0x00de | Kernel already active. | An attempt was made to install a second copy of the kernel Illegal operation. Only one copy of the kernel can run. |
| 0x00e4 | Volume ID is missing. | This is a bug in the user’s application when attempting to use the VIM interface directly. Debug the application. |
| 0x00e5 | Attempt to rename files across volumes. | This operation is illegal. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x00e7 | Can't assign optical drive. Invalid drive ID was specified. | This error message is usually generated for one of three reasons: when a request is made for a drive that is not available, when a drive ID is specified that doesn't exist, or when a drive cannot be used for the requested operation.<br><br>The drive status may be set to off-line. Check the drive status. |
| 0x00f0 | Bad file pointer adjustment. | An attempt was made to illegally change file pointer location to before the beginning of the file. This indicates a bug in the user's applications. This operation is illegal. |
| 0x00fe | File is in the process of being deleted. | This file logically no longer exists. |
| 0x00ff | End of file error. | The control record is missing data segment descriptors which describe the file. Generally indicates that the control record is corrupt. Sometimes occurs if cache files are corrupt.<br>Generally caused by a media error.<br>Check the log file.<br>Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |
| 0x8001 | Operating system independent layer out of memory. | You may not have a large enough page file for the operating system. |
| 0x8002 | Operating system independent layer cannot set file handle count. | Report this error to Pegasus. |

## Memory Manager Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0100 | Memory blocks destroyed. Memory manager control chain has been destroyed. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0101 | Insufficient memory error. | Cannot process request because there is not enough memory. You are probably running the system on the same drive as the operating system swapper. You will need to reconfigure the system so that the operating system virtual memory manager does not fail. This is a system configuration issue. Do not report this bug to Pegasus. |
| 0x0102 | Invalid memory block address. Attempt to free memory not allocated by memory manager. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0103 | Memory failure. General failure of the memory management system. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0104 | Memory has already been freed. Attempt to free memory which has already been freed. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0105 | Null pointer error. A function call was passed a NULL pointer. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

# Btree Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0110 | The btree manager has not been initialized. | You must allow the system to initialize before accessing it. |
| 0x0111 | Attempt to access a cache file that has not been opened. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0112 | Attempt to open a cache file which is already open. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0113 | An invalid cache file control block was passed to a function. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0114 | An illegal btree operation was attempted. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0115 | A function was passed an invalid cache record address. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0116 | Attempts were made to access a cache record that is not active. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0117 | Attempts were made to store a cache record that is too large. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0118 | Attempt was made to delete a cache record which is still active. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0119 | Error traversing cache file free space list. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x011a | Cache file record header is invalid. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x011b | Buffer passed in is too small to read cache record. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x011c | A btree function was passed a NULL record address. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x011d | The cache file free space list has been corrupted. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x011e | A btree index which should exist cannot be found. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x011f | Attempt was made to store a btree index which is too large. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0120 | The size of cache file record is not valid. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0121 | The btree node read does not match the key index. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0122 | Invalid flag found in btree node. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0123 | A compress key in the index file is corrupt. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0124 | A btree node was encountered that was too large or an attempt was made to create a cache file with too large a node size. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0125 | They are no more cache file control blocks available. | There are too many concurrent requests against the system. Retry the operation. |
| 0x0126 | An attempt was made to store a duplicate index. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |
| 0x0127 | An attempt was made to expand a compressed index but the key did not exist. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0128 | Index key comparison failed. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x0129 | An empty btree leaf node was found. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x012a | An operation was attempted on a NULL btree node. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x012b | A btree node has been corrupted. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x012c | A cache file has been corrupted. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |
| 0x012d | Retry key insert - a duplicate key was found. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0140 | Invalid link in cache link list. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0141 | Number of bytes requested to read from cache is bad. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Delete the cache file and restart the system. |

## File Allocation Descriptor (FAD) BTREE Errors

These are generally internal error codes and are not returned to the user

| Error Code Name | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0701 BTCANTOPEN | Can't open FAD file | File may have been deleted underneath system by user. Files should not be deleted in the Pegasus directory unless directed to do so by Pegasus Support |

| Error Code Name | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0702 BTCANTSEEK | FAD file seek failed | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0703 BTCANTREAD | FAD file read failed | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0704 BTCANTWRITE | FAD file write failed | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0705 BTNOSPACE | FAD file disk out of memory | Not enough disk space. Shutdown system updates system to include more disk space. Update Pegasus installation to use added disk space. It is always recommended that the Pegasus be used with a dedicated hard disk to ensure sufficient disk space. |
| 0x0706 BTDUPLICATE | Attempt to write duplicate FAD | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0707 BTUNKNOWN | FAD file corruption | This can occur based on power surge or interruption during operations. It is recommended that the Pegasus server always be run with an Uninterruptible Power Supply (UPS) and formatted with NTFS so that file corruption can be prevented. This can also be the sign of an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0708 BTNOTDONE | FAD function not yet implemented | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0709 BTMATCH | Internal state code | |

| Error Code Name | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x070a BTNONMATCH | Internal state code | |
| 0x070b BTCORRUPT | FAD file corrupt | This can occur based on power surge or interruption during operations. It is recommended that the Pegasus server always be run with an Uninterruptible Power Supply (UPS) and formatted with NTFS so that file corruption can be prevented. This can also be the sign of an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x070c BTBADPARAMETERS | Bad call to FAD routine. | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x070d BTPAGESPLIT | Internal state code | |
| 0x070e BTEOF | FAD search has reached end of file | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x070f BTKEYNOTFOUND | Given FAD key not in database | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |
| 0x0710 BTPAGEDELETED | Internal state code only | |
| 0x0711 BTMEMORYERR | FAD memory access failed | This is an internal error. Please report this error to Pegasus as described in the User's Manual, Appendix B, "Troubleshooting". |

## PFM error codes

| Error Code/ Name | Error Condition | Comment |
|---|---|---|
| 0x801 PFM_FLUSH | A write triggered a flush to optical | Internal code only |
| 0x802 PFM_PARTIAL_FLUSH | A write triggered a partial flush retry | Internal code only |

## Mount Error

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0xffff | Premature end-of-directory list was encountered during a mount procedure. | Generally, this is caused by a media error. Check the log file. Make sure your cartridges and drive have been serviced and cleaned. If the log file reports any type of media error, call the drive or media manufacturer and provide them with the SCSI error codes that have been recorded in the log file. |

# OHIM Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0022 | An operation was specified for an unknown or non-existent disk volume. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |
| 0x0023 | No disk cartridge was found in the drive. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |
| 0x0024 | The export slot already contains a cartridge. No other cartridges can be exported. | You must remove the disk in the export slot before attempting to unmount another cartridge. |
| 0x0025 | Import slot is empty. | You cannot mount the cartridge since it does not exist. You must place a cartridge in the jukebox import slot before it can be mounted. |
| 0x0026 | Jukebox container is full. | You cannot import any other cartridges. You will have to unmount a cartridge before you can import another disk. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0027 | Unable to find any supported SCSI host adapters. | You have either improperly installed the adapter device drivers or you are attempting to use an adapter that is not supported by Pegasus. This is a configuration issue that you must resolve. |
| 0x0028 | Unable to find any support optical drives. | An attempt was made to operate Pegasus software using unsupported hardware. This can indicate that you are attempting to access hardware with the improper Pegasus software model number. |
| 0x0029 | Attempt to import a duplicate disk volume. | User error. This operation is illegal. |
| 0x002A | Error trying to obtain element status from jukebox | The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Could be a hardware problem. |
| 0x002B | The jukebox storage slot is empty. | A request was made to get a disk from an empty storage slot. Generally indicates someone manually removed disks from the jukebox. This operation cannot be supported by Pegasus. DO NOT TOUCH OR REARRANGE CARTRIDGES WHILE THE PEGASUS SOFTWARE IS ACTIVE. |
| 0x002C | Storage slot is full. | An attempt was made to store a disk in a slot that was already occupied. Generally indicates someone manually added a disk to the jukebox. This operation cannot be supported by Pegasus. DO NOT PLAY WITH OR REARRANGE CARTRIDGES WHILE THE PEGASUS SOFTWARE IS ACTIVE. |
| 0x002E | SCSI device is busy. | An attempt was made to send a command to a SCSI device. The device was unable to process the command because it was already busy processing a command. Retry the command. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x002F | A hardware error was encountered while accessing a SCSI device | The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware problems. |
| 0x0030 | SCSI device was not ready. | A command was issued to a device which could not accept it because it was not in a state of readiness. If the system is operating normally, this error can be ignored because it is informational only. If the system is not operational it should be serviced by an experienced technician. |
| 0x0031 | General Media error. | A media error was encountered while attempting to read or write a sector. The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. |
| 0x0032 | Unit attention error. | The device could not accept the command because the state of the device has changed. This often requires operator intervention to correct. |
| 0x0033 | General command abort error. | The SCSI command was aborted due to an unrecoverable error. retry operation. If the error persists, please have the system serviced by an experienced technician. |
| 0x0034 | An illegal SCSI command was issued. | This generally indicates that the incorrect driver was used for a device. It can also indicate that the optical disk contains corrupted pointers. |
| 0x0035 | Unexpected drive reset. | The drive has unexpectedly reset itself. This generally indicates faulty or improper SCSI cabling. This generally indicates there are physical problems with your hardware installation. Please have your system serviced by an experienced technician. Pegasus cannot solve this problem. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0037 | The drive has gone off-line. | The drive is bad and should be replaced. The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware problems. |
| 0x0038 | A disk is in the optical drive but it is not spun up. | Generally, this error is informational only. If the disk cannot be spun up, the system should be serviced by an experienced technician. |
| 0x0039 | General SCSI bus error. | This can be caused by bad or improper cabling, host adapter fault, etc. This generally indicates there are physical problems with your hardware installation. Please have your system serviced by an experienced technician. Pegasus cannot solve this problem. |
| 0x003A | SCSI command was aborted by the host adapter. | Retry command. If error persists, contact your SCSI card manufacturer and provide the errors that have been recorded in the log file. |
| 0x003B | Illegal ASPI was issued by OHIM driver. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |
| 0x003C | Invalid SCSI host adapter ID was sent. | Either the OHIM driver is in error or the adapter card is incorrectly configured. Please have an experienced technician validate the settings of your SCSI card. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x003D | ASPI device selection time-out. The target SCSI device did not respond during the device selection phase. | This generally indicates there are physical problems with your hardware installation. Please have your system serviced by an experienced technician. |
| 0x003E | DMA overrun or underrun. More or less data was received during a data transfer than expected. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x003F | General ASPI SCSI bus error. The host adapter detected a general bus error. | Contact your SCSI card manufacturer and provide the errors that have been logged in the log file. |
| 0x0040 | The jukebox specified is off-line and requires attention. | Have the jukebox serviced. The Pegasus software optical system hardware cannot correct this error. |
| 0x0042 | General device time out. Device will not respond. | This indicates a hardware problem.<br>Have your system serviced. The Pegasus software optical system cannot correct this error. |
| 0x0043 | Jukebox access door is open. | Information error reporting only. Do not report this error to Pegasus. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0044 | Seek error occurred. | This is a hardware error. The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware problems. |
| 0x0045 | Unrecoverable write error occurred on disk. | This is a hardware error. The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware problems. |
| 0x0046 | Unrecoverable read error. This is a hardware error. | The error will be recorded in the log file. Call your drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware problems. |
| 0x0047 | Illegal sector address specified in SCSI command. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0048 | Illegal field in SCSI command. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0049 | Illegal LUN specified in SCSI command. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x004a | Illegal parameter specified in SCSI command. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x004b | Media is write protected. Cannot write to media. | If you want to write to the media, you will need to unmount disk, set protect switch to allow writes and then remount disk. |
| 0x004c | An attempt was made to read a virgin sector. | Generally, this occurs during a mount if the directory area has been damaged. Check the log file for media errors. If they exist, call your media/drive manufacturer with the logged SCSI error codes. |
| 0x004d | An attempt was made to write to a non-virgin sector. | The disk contains unexpected written sectors. |
| 0x004e | Invalid media found in drive. | The media is not supported by the hardware. Remove disks. Do not report this error to Pegasus. |
| 0x004f | Media format corrupted. | The disk must be formatted or reformatted by Pegasus. A disk cannot be used until it is formatted. |
| 0x0050 | No alternate sectors left. | Pegasus will perform its own alternation if possible. The error will be recorded in the log file.<br>Call your media/drive manufacturer and provide the error codes that have been logged in the log file. This indicates severe disk errors. |
| 0x0051 | Fatal ECC error. | Call your media/drive manufacturer and provide the error codes that have been logged in the log file. This indicates severe disk errors. |
| 0x0052 | Could not load/unload media. | This is a hardware error. The error will be recorded in the log file.<br>Call your media/drive manufacturer and provide the error codes that have been logged in the log file. Pegasus cannot solve hardware errors. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0053 | Sector did not verify. Data will be rewritten. | This is an informational error. If a large number of these errors occurs, there is a problem with your system. Either the media or the drive has problems. The error will be recorded in the log file.<br>Call your media/drive manufacturer and provide the error codes that have been logged in the log file. |
| 0x0054 | Error creating semaphore for OHIM. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting."<br>You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0055 | Error requesting OHIM semaphore. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting."<br>You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0056 | Error releasing semaphore. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0057 | Could not allocate memory for OHIM operation. | This indicates that the NT virtual memory manager is failing. You are probably running the Pegasus software on the same drive which is used by the swapper. This cannot be supported.<br>Reconfigure the system to allow Pegasus to run on a different drive or partition. Make sure there is sufficient free disk space for the swap file. |
| 0x0058 | Could not create mail thread. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0059 | You have incorrectly configured the ASPI driver in the Registry. | An ASPI driver is missing. Call your adapter company, or Microsoft for support. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x005c | The OHIM modules have not been initialized. | You must initialize the system and allow it to finish initialization before attempting to access it. |
| 0x005d | Unknown, undocumented error has occurred. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore. |
| 0x005e | Error occurred during IOCTL call to low-level driver. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting.” You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x005f | The low-level driver encountered a fatal error. | This is an internal error. Please report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting." You will need to provide a listing of the log file and with your NT Registry, a complete inventory of your system (what type of computer, how much RAM, hard drive size, and free space available), optical hardware being used, and a directory listing of the \PDT, \PDT\BIN, \PDT\DLL directories or other directories where you installed InveStore.<br>Shut down the system and restart. |
| 0x0060 | Drive hardware error, will retry command and see if problem has been corrected. | Information error report only. Do not report this error to Pegasus. |
| 0x0061 | Media changed. | The drive is indicating a unit attention error that cannot be corrected. Call your drive manufacturer for service. |
| 0x0062 | Mode select changed. | The drive is indicating a unit attention error that cannot be corrected. Call your drive manufacturer for service. |
| 0x0063 | Media unusable. | A piece of media that is not hardware compatible has been inserted into the drive or jukebox. Remove the disk. Do not report this error to Pegasus. |
| 0x0065 | DLL load failure. | The appropriate driver DLL is missing.<br>Check the LIBPATH statement.<br>Reinstall the software. |
| 0x0068 | OHIM error. | The OHIM received an error while sending a SCSI command but was able to correct the error.<br>Make sure your hardware is in good working order. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x0069 | Format failure. | The format command failed while formatting a disk.<br>Replace the disk or have the drive serviced. |

## DLL Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x7001 | Cannot load module. | A DLL cannot be found. The DLL may be missing, or you have an incorrect product.<br>Reinstall InveStore. |
| 0x7002 | Cannot get procedure address. | You may have files that belong to different versions of the software.<br>Reinstall InveStore. |

## Basic Disk Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x9001 | Cannot set default disk. | This error indicates that the InveStore software is having trouble setting up magnetic cache directories. This indicates serious hard disk problems. Have your hard disk replaced or serviced. |
| 0x9002 | Cannot create directory. | This error indicates that the InveStore software is having trouble setting up magnetic cache directories. This indicates serious hard disk problems. Have your hard disk replaced or serviced. |
| 0x9003 | Cannot get current directory. | This error indicates that the InveStore software is having trouble setting up magnetic cache directories. This indicates serious hard disk problems. Have your hard disk replaced or serviced. |
| 0x9004 | Cannot set current directory. | This error indicates that the InveStore software is having trouble setting up magnetic cache directories. This indicates serious hard disk problems. Have your hard disk replaced or serviced. |
| 0x9005 | Cannot get current disk. | This error indicates that the InveStore software is having trouble setting up magnetic cache directories. This indicates serious hard disk problems. Have your hard disk replaced or serviced. |

## Asynchronous Timer Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x1001 | Cannot set async timer. | Report this error to Pegasus. |
| 0x1002 | Cannot set async semaphore. | Report this error to Pegasus. |

## Thread/Process Errors

All thread/process error codes indicate NT resource problems. You may need to modify your setup. For more information, see your Windows NT documentation. The error codes are described below.

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x2001 | Cannot create thread | Modify your setup. Report this error to Pegasus. |
| 0x2002 | Thread cannot initialize. | Modify your setup. Report this error to Pegasus. |
| 0x2003 | Cannot free thread index. | Modify your setup. Report this error to Pegasus. |
| 0x2004 | Cannot set thread index. | Modify your setup. Report this error to Pegasus. |
| 0x2005 | Cannot suspend thread. | Modify your setup. Report this error to Pegasus. |
| 0x2006 | Cannot resume thread. | Modify your setup. Report this error to Pegasus. |
| 0x2007 | Cannot get process ID. | Modify your setup. Report this error to Pegasus. |
| 0x2008 | Thread cannot set priority. | Modify your setup. Report this error to Pegasus. |
| 0x2009 | Unknown thread priority. | Modify your setup. Report this error to Pegasus. |
| 0x2010 | Cannot set priority. | Modify your setup. Report this error to Pegasus. |

# Semaphore Manager Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x3001 | Semaphore timeout. | Report this error to Pegasus. |
| 0x3002 | Cannot create semaphore. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3003 | Cannot close semaphore. | This error may indicate NT resource problems. Report this error to Pegasus. |
| 0x3004 | Cannot release semaphore. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3005 | Cannot duplicate semaphore. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3006 | Cannot open semaphore. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3007 | Semaphore not supported. | This is an internal error.Report this error to Pegasus. |
| 0x3008 | Cannot lock semaphore. | This is an internal error.Report this error to Pegasus. |
| 0x3009 | Cannot enter critical section. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3010 | Cannot request mutex. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3011 | Cannot wait for event. | Report this error to Pegasus. |
| 0x3012 | Cannot leave critical section. | Report this error to Pegasus. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x3013 | Cannot create event. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3014 | Cannot close event. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3015 | Cannot open event. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3016 | Cannot clear event. | This error may indicate NT resource problems. Modify your setup. Report this error to Pegasus. |
| 0x3017 | Cannot reset event. | This is an internal error. Report this error to Pegasus. |

# HCM Errors

| Error Code/Name | Explanation/Possible Fix |
|---|---|
| 0x0501 ERRHCM_RAM_CACHE_ EXISTS | HCMinitCache was called to create a RAM cache, but the RAM cache already exists. Only one RAM cache is supported. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0502 ERRHCM_DASD_CACHE_ EXISTS | HCMinitCache was called to create a DASD cache, but the DASD cache already exists. Only one DASD cache is supported. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, “Troubleshooting” the InveStore software. |
| 0x0503 ERRHCM_INVALID_CACHE_FI LE | The DASD cache is in an invalid state and cannot be recovered. The system will need to be redone. Repair or replace hard disk. Cache file will need to be deleted. Make sure system is properly functioning. |
| 0x0504 ERRHCM_INVALID_FUNCTION | An HCM function was requested which is either unsupported or which is sometimes supported but was called in an incorrect context. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting” the InveStore software. |
| 0x0505 ERRHCM_INVALID_ PARAMETER | An HCM function was called with an invalid argument. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting” the InveStore software. |

| Error Code/Name | Explanation/Possible Fix |
|---|---|
| 0x0506<br>ERRHCM_INVALID_VOLUME_ID | An HCM function was requested for a Virtual Volume which has not been registered with the HCM. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0507<br>ERRHCM_INVALID_VOLUME_REC | The volume record in the DASD cache is not consistent with the volume record in the RAM cache. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0508<br>ERRHCM_ACCESS_COUNT_NONZERO | A volume was not released after a request to HCMreleaseVolume due to an error in the access count. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0509<br>ERRHCM_ACCESS_COUNT_OVERFLOW | Too many recursive accesses have been performed on a volume. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x050a<br>ERRHCM_ACCESS_COUNT_UNDERFLOW | Access to a volume has been released more times than it was requested. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |

| Error Code/Name | Explanation/Possible Fix |
|---|---|
| 0x050b ERRHCM_DUPLICATE_ VOLUME_ID | An attempt was made to register a Virtual Volume which is already registered with the HCM. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x050c ERRHCM_CANNOT_ALLOCATE _RAM_NODE | The memory allocator for HCM RAM cache is out of space and is unable to free space. Indicates that the RAM cache should be larger. Check size of page file. Increase physical RAM. Increase size of InveStore RAM cache. |
| 0x050d ERRHCM_CANNOT_ ALLOCATE_DASD_NODE | The memory allocator for HCM DASD cache is out of space and is unable to free space. Indicates that the cache should be larger. Check free space. It is always recommend that the Pegasus cache be given a dedicated hard drive. |
| 0x050e ERRHCM_CANNOT_ ALLOCATE_DASD_BUFF | The memory allocator for HCM DASD cache is out of space and is unable to free space. Check free space. It is always recommend that the Pegasus cache be given a dedicated hard drive. |
| 0x050f ERRHCM_SECTOR_NOT_ FOUND | A request was made to locate a sector in the HCM, but it was not present in the cache. This is an internal error code |
| 0x0510 ERRHCM_NULL_BUFFER_ PTR | A buffer descriptor exists in the HCM for some sector, but no data buffer exists. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |

| Error Code/Name | Explanation/Possible Fix |
|---|---|
| 0x0511 ERRHCM_GEN_FAILURE | Some condition has arisen which the HCM is unable to deal with, either because the condition should never occur, or because the HCM is not properly programmed to deal with it. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0512 ERRHCM_DASD_ UNSUPPORTED | A request to the HCM DASD manager was made in a system for which the DASD cache is not implemented. Hard disk caching is not currently supported. |
| 0x0513 ERRHCM_VOLUME_ RESERVED | A request for volume access must wait because the volume is reserved by another thread. This is used only within HCM. Internal only |
| 0x0514 ERRHCM_VOLUME_OFFLINE | An attempt was made to access a volume which is not mounted. This is typically a user or application error. Mount volume or redirect the request to an online volume. |
| 0x0515 ERRHCM_READ_FAULT | Some condition has arisen while reading which the HCM is unable to deal with. A data error occurred during writing to optical. Check the log file for optical hardware errors. This problem can be a hardware issue. Please contact drive manufacturer. Pegasus cannot solve hardware issues This can also be caused by a programming deficiency in the HCM or its caller. Then it is an internal error Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |

| Error Code/Name | Explanation/Possible Fix |
|---|---|
| 0x0516<br>ERRHCM_WRITE_FAULT | Some condition has arisen while writing which the HCM is unable to deal with. This is caused by a programming deficiency in the HCM or its caller. This is an internal error. Please Report this to Pegasus following the procedure specified in Appendix B, "Troubleshooting" the InveStore software. |
| 0x0517<br>ERRHCM_WRITE_ERROR | A data error occurred during writing to optical. Check the log file for optical hardware errors. This problem is a hardware issue. Please contact drive manufacturer. Pegasus cannot solve hardware issues. |
| 0x0518<br>ERRHCM_WRITE_ERROR2 | Some error other than a data error occurred during writing to optical. Check the log file for optical hardware errors. This problem is a hardware issue. Please contact drive manufacturer. Pegasus cannot solve hardware issues. |

## Interprocess Communications Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x4001 | Cannot create npipe. | This error may indicate NT resource problems. |
| 0x4002 | Cannot connect npipe. | The kernel software may be down. Verify that the system is running. |
| 0x4003 | Npipe wait failed. | This is an internal error. Report this error to Pegasus. |
| 0x4004 | Cannot disconnect npipe. | This is an internal error. Report this error to Pegasus. |

## File Errors

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x6001 | Bad open flags. | Debug your application. |
| 0x6002 | Bad attribute flags. | Debug your application. |
| 0x6003 | File open failed. | An expected cache file does not exist. This may also be an internal error. Report this error to Pegasus. |
| 0x6004 | File read failed. | Debug your application. Report this error to Pegasus. |
| 0x6005 | File write failed. | Debug your application. Report this error to Pegasus. |
| 0x6006 | File close failed. | Debug your application. Report this error to Pegasus. |
| 0x6007 | File delete failed. | Debug your application. Report this error to Pegasus. |
| 0x6008 | Cannot set file pointer. | Debug your application. Report this error to Pegasus. |

## Shared Memory Errors

The shared memory error codes described below typically indicate NT resource problems. The page file may not be large enough. It is recommended that you pre-allocate a 100M byte page file for the NT system and that you *not* depend on having the page file grow dynamically. For more information, see your Windows NT documentation.

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x5001 | Cannot create mutex. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5002 | Cannot create file mapping. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5003 | Cannot map view of file. | This error may indicate NT resource problems. Reallocate the page file size. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x5004 | Cannot initialize shared memory. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5005 | Cannot open shared memory. | Report this error to Pegasus. |
| 0x5006 | Bad allocation size. | This is an internal error. Report this error to Pegasus. |
| 0x5007 | Cannot allocate shared memory. | This is an internal error. Report this error to Pegasus. |
| 0x5008 | Cannot give shared memory. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5009 | Bad free size. | This is an internal error. Report this error to Pegasus. |
| 0x5010 | Cannot free shared memory. | This is an internal error. Report this error to Pegasus. |
| 0x5011 | Cannot close shared memory. | This is an internal error. Report this error to Pegasus. |
| 0x5012 | Cannot allocate memory. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5013 | Cannot free memory. | This is an internal error. Report this error to Pegasus. |
| 0x5014 | Cannot allocate shared memory mapped file. | This error may indicate an NT resource problems. Reallocate the page file size. |
| 0x5015 | Cannot create shared memory file mapping. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5016 | Cannot map view of shared memory file. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5017 | Cannot create shared memory file. | This error may indicate NT resource problems. Reallocate the page file size. |

| Error Code | Error Condition | Explanation/Possible Fix |
|---|---|---|
| 0x5018 | Cannot unmap view of shared memory file. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5019 | Cannot close shared memory file. | This error may indicate NT resource problems. Reallocate the page file size. |
| 0x5020 | Cannot free shared memory file. | This is an internal error. Report this error to Pegasus. |
| 0x5021 | Cannot close shared memory file. | This is an internal error. Report this error to Pegasus. |
| 0x5022 | Cannot open file mapping. | This error may indicate NT resource problems. Reallocate the page file size. Report this error to Pegasus. |
| 0x5023 | Cannot get named memory. | This error may indicate NT resource problems. Reallocate the page file size. |

# Appendix B

## TROUBLESHOOTING

This appendix provides information on InveStore troubleshooting. Please review the information below before calling Pegasus Technical Support.

Many of the problems experienced during installation and initialization can be resolved by simply reviewing the details contained within this manual along with the list of commonly encountered problems and associated solutions provided below.

For general support related questions visit our online eSupport search engine at http://www.pegasus-ofs.com/eSupport. For API questions visit http://www.pegasus-ofs.com/apisupport.

For further assistance, please contact us directly as outlined in the following section.

# Have Your Information Ready

Pegasus provides a special utility that collects information about your system configuration. To use the utility, open a DOS window and perform the following procedure:

At a DOS command prompt enter **PSUPPORT.BAT**

When PSUPPORT gathers all the information about your system, follow the instructions on your screen to place the information into a file called PSUPPORT.TXT.

To get help by email, you can email a description of your problem to the following address:

support@pegasus-ofs.com

When you call Pegasus for technical support, have the following:

- A printout of the PSUPPORT.TXT file
- Log file output from the day/time error occurred. The log files are time/date stamped and can be found in the PDT\BIN directory or, if you installed files in a different subdirectory, go to that location. The log file names are OJ*xxxxxx*.LOG and KLxxx.LOG, where *x* is the current date.

You will be asked to provide the following information:

- A listing of the log file along with a printout of your NT registry.
- A complete inventory of your system, including what type of computer you have, size of RAM and hard disk, available free space, and size of the page file.
- Optical hardware you're using.
- A directory listing of the \PDT, PDT\BIN, and \PDT\DLL directories if you installed the software in the default location.

# Frequently Asked Questions

The most commonly asked questions and answers to them are:

**1** Is making direct API calls faster than if we just read and write to the drive letter assigned to Pegasus?

**Answer:** Yes. Using the API eliminates several transitions from User mode to kernel mode and back. It also eliminates all the code that is required to massage information sent and received from the API into the form required by the operating system; this code is substantial.

As an added benefit, you will gain more control over the entire system. You will receive error messages that are far more meaningful than those that are mangled to match error codes supported by the operating system. You will be able to access calls that cannot be used transparently, such as Mount, Unmount and Format. You will also be able to obtain more detailed information about volumes, files and subdirectories.

**2** Is it possible to perform all jukebox operations without the GUI running?

**Answer:** Yes the GUI uses the VIM API exclusively. All functions contained within the GUI directly correlate to VIM functions. The only exception to this is the error logging. The GUI uses an undocumented IPC messaging protocol to receive log messages. The kernel, under version 3 will continue to log messages to a separate file if the GUI does not register itself as the system console. This log file is then available to the user.

**3** What version of "C" was InveStore coded with?

**Answer:** InveStore Release 3 for NT was coded using Microsoft Visual C version 4.2. This is subject to change with future releases.

**4** Do you support Visual Basic?

**Answer:** Not directly but you can use Visual Basic to access the VIM API. The API has been changed to use the _stdcall calling convention which is supported by Visual Basic. You will need to declare the VIM functions as external function calls, as shown in the following examples.

## Example: Declaring VIM_set_drive_bias

The following example shows how you can use the convention for calling the VIM function VIM_set_drive_bias:

```
Declare Function VIM_set_drive_bias Lib "vim32.dll"
   (ByVal DeviceId As Integer, ByVal bias As Long)
   As Long
```

### Example: Declaring VIM_disk_params

Here is another example of a more complicated API Declaration in Visual Basic, showing an example of how to declare VIM_disk_params:

```
Type CurDirInfo
    subdir_id As Long
    volume_id As Integer
    reserved  As Long
    reserved2 As Integer
End Type

Declare Function VIM_disk_params Lib "vim32.dll"
  ( ByVal volume_name As String, cur_directory
  As CurDirInfo, sector_size As Integer,
  total_sectors As Long, free_sectors As Long )
As Long
```

Note: In VB, Integer is 2 bytes and Long is 4 bytes. In the native Pegasus 32-bit environment, integer and long are the same size 4 bytes.

### Example: Declaring VIM_format

```
Declare Function VIM_format Lib "vim32.dll"
  (ByVal DeviceID as Integer, ByVal volume_name
  As String, ByVal TrappedName As String,
  ByVal MountAfterFormat As Long ) As Long
```

### Example: Declaring VIM_unmount

```
Declare Function VIM_unmount Lib "vim32.dll"
  (ByVal volume_name As String, ByVal UnmountType
  as Long, ByVal DeviceID as Integer, ByVal
  OfflineLocation As String) As Long
```

### Example: Declaring VIM_mount

```
Declare Function VIM_mount Lib "vim32.dll"
(ByVal volume_name As String, ByVal DeviceID
as Integer, ByVal MountType as Long,
ByVal ByVal FileSystemType As Long) As Long
```

**5** Can I directly call the OHIM Layer and bypass InveStore?

**Answer:** The OHIM API has not been documented. While this API is directly callable it requires understanding how each function is used and

when it may be called. This layer may be documented in a future release of the programmers toolkit.

**6** I'm using the VIM API and your software crashes or returns bad results. Does this mean the API call has a bug?

**Answer:** While it may be possible that you have uncovered a bug it is not very likely. If you pass in bad parameters the Pegasus software is not protected from crashing. All GUI functions and all transparent OS calls go through the VIM API. Operating System functions such as "DIR", "XCOPY", "ATTRIB", "COPY", and so forth must go through the VIM API. The VIM API has been heavily tested for years. In most cases, bad results from a VIM call or having the Pegasus software crash when using a VIM call is the result of improper VIM API usage.

**7** Does the GUI really use VIM API calls?

**Answer:** Yes, the GUI uses the following API calls:

VIM_search()
VIM_get_volume_info()
VIM_get_hardware_list()
VIMgetHardwareKistSize()
VIM_set_drive_status()
VIM_set_drive_bias()
VIM_get_drive_bias()
VIM_mount()
VIM_unmount()
VIM_format()
VIM_format_media()
VIM_rename()
VIM_shutdown()
VIM_commit_volume()

For more information, see "Map of VIM Calls to GUI Features" on page 12.

**8** How do I know whether a VOLUME is online or offline and which library it is in?

**Answer:** The OHIMslotID in the volume record will be set to some other value than 0xffff. For more information on volume record, see "Volume Record" on page 31. All offline volumes do *not* have a valid OHIMslotID. This OHIMslotID tells you which library contains the volume. You call VIM_get_hardware_list and look into the LirbaryStatus array. You find which Library contains this volume by

checking the MinimumVol and MaximumVol fields. The OHIMslotID will be greater than or equal to MinimumVol, and less than or equal to MaximumVol for the Library that owns this volume.

**9** How Do I control which drive a volume goes into?

**Answer:** You can't. This is the responsibility of Pegasus software. The drive ID in any VIM calls is used only to define for which Library the call is intended. It will use the drive specified, if possible, but on most systems this rarely occurs.

**10** Can I tell which drive a volume is in?

**Answer:** Yes, the VolStatus array off the LibraryStatus array will tell you exactly where the volume resides, either in a drive or in the storage slot. For more information, see "VIM_get_hardware_list" on page 78.

**11** How can I tell what storage slot a volume is in?

**Answer:** The OHIMslotID in the volume record defines what storage slot the volume resides in. For more information on volume record, see "Volume Record" on page 31. For double-sided media, there are two OHIMSlotIDs for each disk. Single-sided media has one OHIMslot ID for each. Each Library has a range of OHIMslotIDs assigned to it. OHIMslotIDs are in sequential ascending order. To determine the actual slot subtract the Base OHIM slot ID for the Library from the given ID and divide by 2 if double sided media.

**12** How can I force a disk to be removed from a drive?

**Answer:** You can't. InveStore runs in a multitasking, multithreaded environment. Only InveStore has enough information to know when it is appropriate to both remove disk from a drive, and place a disk into a drive.

**13** What is the VirtualVID of the VolumeRecord and how does it differ from the OHIMslotID?

**Answer:** The VirtualVID is a direct access index used for accessing volume records in the volume database. The OHIMslotID defines what shelf is the home storage shelf for an online/nearline volume.

**14** How do I tell what volume was mounted after I issue a VIM_mount command?

**Answer:** You must send a VIM_search command and find out what new volume or pair of volumes has been added to the directory.

**15** I have many disks that I would like to format all at once. How do I do that?

**Answer:** Use the VIM_format routine. Load the disks to be formatted either via the front panel of the jukebox (without InveStore running) or via the VIM_mount command. Your applications should create a separate thread of execution for each disk side that you wish to format at the same time. Select one of the !INI volumes for the "Trapped Volume Name" as the source volume for the format. Under version 3.0, the format command executes simultaneous formats, up to the number of available drives. Any additional format requests are queued up within the Pegasus kernel until a drive becomes available. It is recommended that this type of formatting only be done when InveStore is not available to network clients, since typing up all available drives would make network access impossibly slow.

# Common Problems

The following are common problems. A solution for each is presented.

### Lack of structure packing

You must pack all structures used when calling the VIM API. The /Zp1 option for the compiler must be used. For more information, see “Compiling and Linking” on page 16.

### Using an Unsupported Calling Convention to Access the VIM API

The VIM API uses the _stdcall calling convention. The prototypes in VIMAPI.H enforce this convention. You must indicate to the compiler and linker that the VIM API uses _stdcall. For more information, see “Calling Conventions” on page 26.

### Not including VIM32.LIB in your LINK statement.

The Vim32.LIB is required by the linker to resolve the external references to the VIM API. The linker must be able to find this lib file as well. If the lib file is not included and accessible the linker returns various "Unresolved External References" errors.

### Example of Error

MYCODE.obj: error LNK2001: unresolved external symbol "int __stdcall VIM_get_object_info(char *,struct CurDirInfo *,struct ofs_records *)"(?VIM_get_object_info@@YAHPADPAUCurDirInfo@@PAUofs_records@@@Z)

### No Memory Allocated for Structures, Arrays and Buffers.

You must properly allocate memory when accessing the VIM API. The VIM API does *not* allocate memory for you. Make sure that you provide valid pointers (with enough memory allocated to them) to the VIM API.

### Functions that Do Not Require Memory Allocation

The following calls are the only ones that do not require pointers:

| | |
|---|---|
| VIMcommitVolume | VIM_set_drive_bias |
| VIM_check_eof | VIM_set_drive_status |
| VIM_cleanup_process | VIM_set_time_date |
| VIM_close | VIM_shutdown |
| VIM_flush_buffer | VIM_truncate_file |
| VIM_initialize | |

### Functions that Do Require Memory Allocation

You must allocate buffers of sufficient size for VIM_write, VIM_read, VIM_get_hardware_list.

Also, when passing in an array that contains a volume name, you must make the array large enough to handle the volume name. Future releases will support 256-byte volume names, so we recommended that you chose array names of more than 256 bytes.

### Compiling with Old Library and Header Files

To successfully program to the VIM API, you *must* use the current header files. Using old and out of date header files causes many problems. The header files and example code can be found on the InveStore CD and in the PDT\VIM directory after InveStore is installed.

### Using the wrong API calls

Developers who don't read the manual usually use the wrong API call. You need to carefully read and understand what each VIM call does, especially VIM_format and VIM_format_media, which perform the same operation, but in a very different manner. Understanding the VIM calls is key to using the proper API call and using it correctly.

### During system boot, the computer hangs

Possible causes: This problem may occur because of possible conflicts within the system. By asking and answering the following questions, you can resolve most conflicts:

**1** Are there any interrupt channel conflicts between the SCSI host adapter and any other peripheral card? (System may hang, and no keyboard input is accepted.)

Solution: If two peripheral cards attempt to use the same interrupt channel, problems will occur. Change the interrupt channel to use an available channel.

**2** Are there any I/O port address select conflicts within the system? (System frequently hangs for no apparent reason or occasionally fails to boot.)

Solution: Verify that there are no other peripheral cards present in the system using the same I/O port address.

### You suspect problems between the SCSI card and the hardware:

Check the items listed below:

- Is the SCSI host adapter properly cabled and connected to the one or more optical drives or jukebox? (Please consult the troubleshooting section of the SCSI hardware installation guide for proper installation instructions).
- Are the optical drives or is the jukebox powered on?
- Are the SCSI termination resistors installed on the host adapter?
- Is the fuse on the SCSI host adapter good?
- Is the last SCSI device on the SCSI bus terminated?
- Does each SCSI device have an individual SCSI ID?

All devices MUST have different SCSI IDs. Jukeboxes must have the SCSI IDs set as follows:

Jukebox ID = Lowest available ID, drive 0, next lowest, drive 1, and so on.

Valid SCSI IDs are: 0, 1, 2, 3, 4, 5, 6.

1. Is there a SCSI hard drive present in the system? Most SCSI boot devices are configured as SCSI ID=0. Ensure that additional devices are configured such that they do not conflict with existing installed devices.
2. If there are no SCSI hard drives present, the host adapter BIOS should be disabled.

For additional troubleshooting suggestions, please consult your SCSI hardware installation manuals or call the SCSI host adapter manufacturer for technical support.

### No Supported SCSI Host Adapters Found

Possible causes: If this message appears, the SCSI host adapter installed is NOT supported by InveStore.

Solution: Replace the non-supported SCSI host adapter with a SCSI host adapter that has been fully tested and qualified for use with InveStore. Supported SCSI adapters are listed on the Optical Hardware Supported document. You can download a current list of supported hardware from our web site at http://www.pegasus-ofs.com.

**Note** If you cannot get the SCSI driver to load successfully, call the SCSI board manufacturer. Pegasus cannot resolve this problem.

### Insufficient Magnetic Disk Space -- Disk is Full (errors 0x00c6, 0x00c9)

Message reported to screen or Log file, in the PDT\BIN directory or other directory where you installed InveStore.

Possible causes: This error occurs when the magnetic hard disk containing the PDT subdirectory has less than 6M bytes of disk space available. Pegasus requires a minimum 6M bytes of available disk space. Pegasus creates an optical volume table containing optical disk directory information and records to the magnetic disk drive. To ensure that there is suffi-

cient space available for creation and maintenance of these files, Pegasus checks available magnetic disk space when InveStore is initialized.

Solution: Make more disk space available. Delete all unnecessary files from the hard drive, or add a larger capacity hard disk.

### Volume in Slot is NOT Pegasus Format and cannot be Referenced (error 0x00af)

Possible causes: This message appears when a user attempts to mount, read or write to a non-Pegasus formatted disk, or when a user attempts to re-initialize a previously used WORM disk which has been used with another file system.

Solution: WORM media is (W)rite (O)nce, (R)ead (M)ultiple media. Once initialized by one file system, it is unusable with another system. Replace the disk with a new disk that has not been written to.

### Drive Not Ready

Possible causes: This message may appear for many possible conditions, including:

**1** Drive busy

**2** Drive not ready

**3** Disk not in drive

**4** Disk is not spinning up as expected

**5** Drive is having problems reading a disk (dirty drive, media, slow servo motor)

**6** Bad or improper SCSI bus termination

### Access Denied

Possible causes: This message occurs for several reasons:

**1** The volume request is not mounted, and the retry time-out has expired.

Solution: Mount the requested volume.

**2** All available drives are busy/locked (i.e., disks with open files).

Solution: Retry the operation until a drive becomes available.

**3** The file is read only

**4** The volume requested is write protected.

### Insufficient Memory (swap drive space)

Possible causes: This message appears when the user has exceeded the amount of available system memory.

This indicates that the drive used by the NT pagefile is full. More space should be made available on the swap drive.

### Magnetic File Open Failure

Possible causes: This message is usually encountered when the amount of available system memory has been exceeded, when someone has inadvertently deleted cache files while the system was running, or when there is a physical problem reading cache files on the hard drive. Run CHKDSK and DEFRAG.

### Invalid System Parameter

Possible causes: This message appears when parameters are changed to invalid settings in the OPTICAL.CNF file located in the PDT directory.

Solution: In the InveStore console window, open the Edit menu and choose Optical Subsystem configuration. Adjust the parameters to appropriate values.

### Error Removing Disk

Possible causes: This error may result from one of the following conditions:

**1** The optical drive did not spin down as expected.

**2** The jukebox move operation exceeded the maximum time-out allowed.

**3** The mailbox door failed to open or could not transport the disk.

### Volume Not Mounted, or Error Mounting Disk

Possible causes: These errors are reported when a disk cannot be successfully read, or when there was a problem encountered writing the magnetic directory cache file.

Solution:

1. Check the PDT error log and identify the associated time and date stamped error message.
2. Inspect the drive/jukebox. Watch it mount a disk. Perform subsystem maintenance as required.

# Index

A
Application Suggestions 38
Attributes 103, 105
B
Bias 61, 62, 101
Bytes Free 20
C
Cache 58
Cache Write Algorithms 36
Calling Convention 175
Calling Conventions 26
Calls 65
commit 17
Committing a Volume 17
Common Fields 28
Common Problems 175
Compile 16
Compiling and Linking 16
Configurable parameters
    Cache 56
        Default maximum cache page 57
        Default minimum cache page 57
        Default write cache algorithm 56
        Maximum ram cache buffers 57, 58
    Disk Swapping 59
        Drive biases 62
        Hold time 60
        Lock time 60
        Off-line drives 61
        Priority wait time 60
        Request mount time 59
        Request wait time 60
        System bias 61
    General 56
        Delete files 56
        Partition 56
        replace empty files 56
        Retry optical errors 56
    Mount 58
        Mount validation 59
        Premount all disks 58
Control Area 47
D
Data Area 47
Directory 29
Directory Structure 27
Disk Swapping Parameters 59
Documentation conventions 9
DVD 49
DVD-ROM 48
E
Error code directory 117
Error log file 116
    Format 116
Errors
    Asynchronous timer 157
    Basic disk 157
    btree 130
    DLL 156
    Fatal 116
    General 117
    Memory manager 128
    Mount 143
    OHIM 144
    Semaphore manager 159
    Shared memory 166
    Thread process 158
F
file 52
File Formats 44
File Record 30

File System 52
Flush On Close 56
Flush on Close 37
Flushing Volumes 18
Format 74
Free Disk Space 20
Free Space 19, 21
H
Handles 56
Hardware 78, 85
HCM 43, 44, 54
Header Files 15, 176
Hierarchical Cache Management 43
I
Identifying Unflushed Volumes 39
Initialization 63
Initialize 74, 89
Installable File System (IFS) 53
InveStore Records 27
K
Kernel 52, 53
Kernel initialization 63
L
Lazy Write 36, 56, 58
Linking 175
Logical File Structure 48
M
Make Directory 90
Mount 90
Mount Parameters 58
O
Off-line Drives 61
OHIM 42, 53, 56
Open 92
OPTICAL.CNF 55
P
Parameters 55, 56, 58, 59, 180
Partition 56
Pathname Rules 36
Pegasus Engineering Services 10
Physical File Structure 46
Problems
    Disk format 179
    Disk removal 180
    Drive not ready 179
    File open 180
    Memory 180
    Mountng 180
    SCSI support 178
    System parameters 180
    Volume access 179
Programmer Support 10
Programming Examples 110
R
Read 97
Record Object Union 35
Remove Directory 99
Rename 98
Return Codes 36
S
SCSI Adapters 178
Search 100
Shutdown 107
Structure Packing 175
Subdirectory Record 31
Supported Calls 23
System Data Types 26
System Parameters 55
T
Terminology 9
Threads 42, 52
Time/Date Stamp 104
Transparent Bytes Free 20
Troubleshooting 17, 169
    Common problems 175
    Pegasus Technical Support 169

U
UDF 46
Unmount 108
Using ATTRIB 18
V
Visual 15
Visual Basic 16, 171
Visual C 171
Visual C/C++ 15
Volume Info 88
Volume Manager 41
Volume Record 31
Volume Types 42
W
Win32 Environment 17
WORM 46, 48, 49
Write 109
Write Through 37, 56